Institut für Parallele und Verteilte Systeme
Abteilung Parallele Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2391

Droplet feature extraction from images obtained from spray drying process

Xiaolei Guo

Course of Study:	Computer Science
Examiner:	Prof. Dr. Sven Simon
Supervisor:	M. Sc. Silvia Ahmed
Commenced:	June 01, 2012
Completed:	December 01, 2012
CR-Classification:	I.3.5, I.4.3, I.4.7

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Motivation

Image processing is not only used in the normal daily life for personal usage, it is also widely used in the industrial area. For example, in pharmaceutical and chemical industries spray processes are used to remove solvents from thermolabile compounds. In order to analyze the formation of morphology and shape forming inside the spray tower an image of a droplet should be taken in the stages and the required geometry from the image should also be analyzed.
In most cases it is not easy to extract the droplet because of its transparency. The difference between the droplet and the background is usually small, thus the normal single image processing methods are usually not appliable. A combination of different processing methods is needed, which can make a good preparation to get the properties of the droplet.

1.2 Aim

The goal is to find a suitable combination of normal image processing methods which can extract the droplet from the background in the image. Due to the features of the image, the normal image processing methods can not be used directly, some specification and modification should also be added.
When the general combination is found, some effort should be given to reduce the requirements of the algorithm. It is better to process the image on the fly, which means that this algorithm is able to process the image even if it doesn't get the full view of the image. The minimal usage of memory and bandwidth will be preferred. The to be processed image in this article is depicted in Figure-1.1:

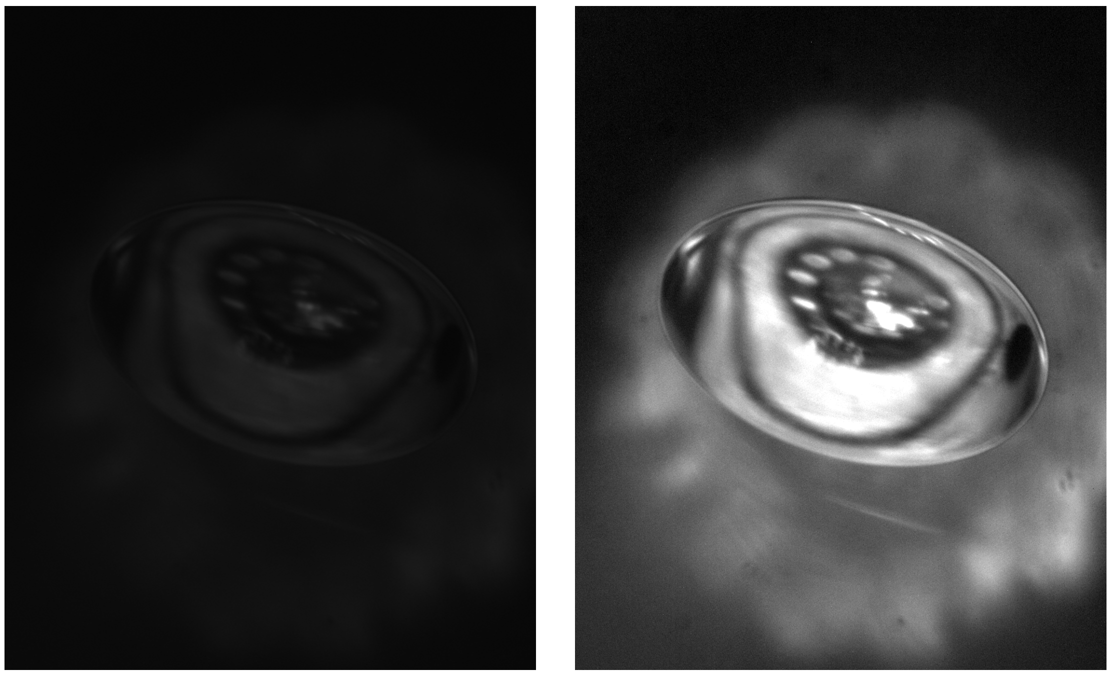

(a) original droplet image **(b)** contrast enhanced image

Figure 1.1: Original Droplet Image

1.2.1 Tasks

- Compare the common used image processing methods and choose a proper composition of them. Use this to extract the droplet object from the image.
- Find methods to extract the features of the droplet, such as axis lengths and thus the area, volume, etc .
- Try to modify the combined methods or algorithms to reduce the memory and bandwidth usage.

2 Pre-Processing Methods

When an image of an object is captured, the image is not always readable for human eyes. Sometimes the boundaries of the object is not clear enough, extra noise pixels may also be added due to the functionality of the image capturing mechanism. In order to reduce the negative effect and do a good preparation for the image property calculation step , image pre-processing methods should be introduced.
In this chapter some useful methods for image processing are discussed.

2.1 Threshold

The pixel value varies both in object and on the edge. When all of the changes are treated as changes on the edge, an image with a lot of useless noise pixels exists. Thus a threshold value is needed to seperate the usefull data from the noise.
There are a lot of threshold value generating methods. The most used one is the global threshold[3] . The main steps of the global threshold value generation is described as follows:

- Collect all values in the computing domain.
- Find the maximum value **MAX** and the minimum value **MIN**.
- Calculate the median **T** by doing $(\mathbf{MAX} + \mathbf{MIN})/2$.
- **T** seperates all values into 2 parts: values bigger or equal to **T**, and the rest.
- Calculate the mean value of the two parts, and set them as **MAX** and **MIN**. Then we will get another median value **T'**.
- If $abs(\mathbf{T}' - \mathbf{T}) < 0.5$, the calculation will end, and **T** is the threshold. If not, T will be set with the value of **T'**, and do another loop will begin from the seperating step.

But this threshold doesn't always work perfectly in the edge detection step. According to the experiment, it is better to lower the value a bit, such as use only 70% of this threshold .
In the next section some edge detection filters will be introduced.

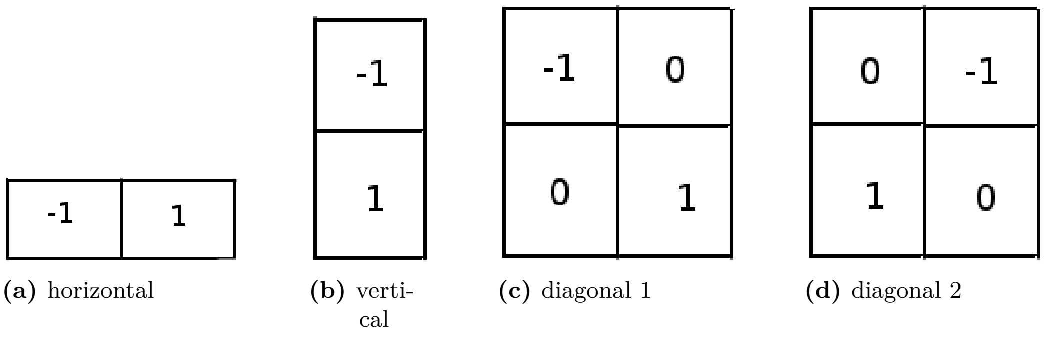


Figure 2.1: Line Filters

2.2 Edge Detection

In order to extract the droplet from the image, a method to find the boundary between the droplet and the background is needed. A basic idea to identify the boundary (or edge) between two object is to check how fast the pixel value varies. The basic assumption is that the acute changes of pixel value most happen at the boundary instead of in the object. the ideal situation is that all pixels on the edge can be found through calculating the derivatives (first-order) of the pixel value , and by connecting these pixels the boundaries can be obtained.

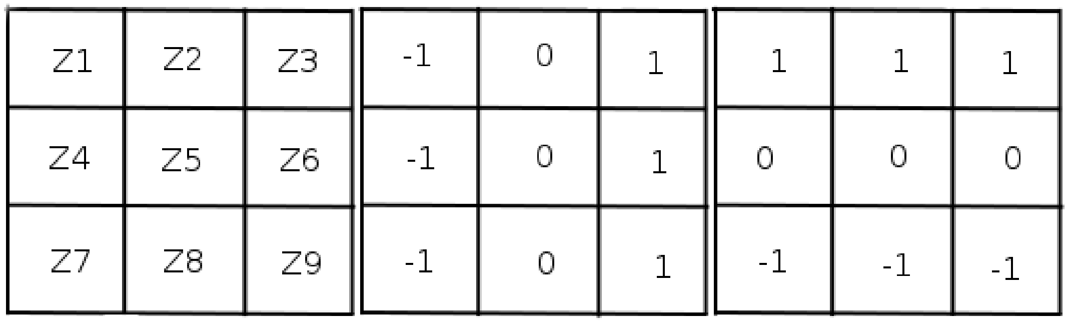
2.2.1 Line Filters

These filters [3] calculate the difference between 2 neighbouring pixels or 2 diagonal neighbouring pixels. 4 directions (or 4 connectivities) can be calculated: up-down, left-to-right, left-top to right-bottom, left-bottom to right-top.(The latter 2 are also called roberts cross gradient operators.)
Figure-2.1 shows the usually used filters.

2.2.2 Prewitt Filter

This method [3] convolves a image with a small integer-valued operator in horizontal or vertical direction. The smallest filter masks are of size 3×3 . When the horizontal approximation g_x and the vertical approximation g_y are calculated , a 3×3 neighbourhood to calculate the center value z_5 is used , which is shown in Figure-2.2.

$$g_x = \frac{\partial f}{\partial x} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$



Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

(a) indices

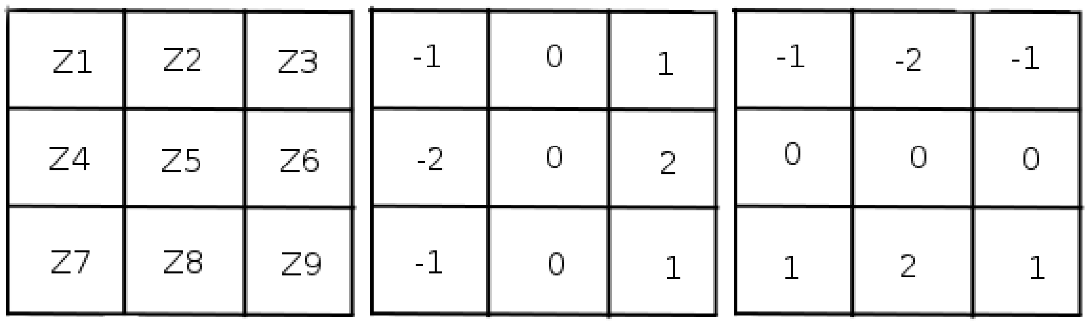
-1	0	1
-1	0	1
-1	0	1

(b) horizontal

1	1	1
0	0	0
-1	-1	-1

(c) vertical

Figure 2.2: Prewitt Filter Masks



Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

(a) indices

-1	0	1
-2	0	2
-1	0	1

(b) horizontal

-1	-2	-1
0	0	0
1	2	1

(c) vertical

Figure 2.3: Sobel Filter Masks

$$g_y = \frac{\partial f}{\partial y} = (z_1 + z_2 + z_3) - (z_7 + z_8 + z_9)$$

With g_x and g_y the gradient magnitude can be calculated by using

$$g = \sqrt{g_x^2 + g_y^2}$$

2.2.3 Sobel Filter

The sobel filter [3] works almost the same as prewitt filter. The only difference is that the sobel filter calculates an extra gradient in horizontal or in vertical direction, just as showed in the masks in Figure-2.3.

$$g_x = \frac{\partial f}{\partial x} = (z_3 + z_6 \times 2 + z_9) - (z_1 + z_4 \times 2 + z_7)$$

$$g_y = \frac{\partial f}{\partial y} = (z_7 + z_2 \times 8 + z_9) - (z_1 + z_2 \times 2 + z_3)$$

$$g = \sqrt{g_x^2 + g_y^2}$$

The effect of sobel filter is similar to the effect of prewitt filter. The difference between the two is that sobel filter concerns more about the neighbouring pixels in the vertical and horizontal direction.

2.2.4 Laplacian of Gaussian Filter

All of the filters above don't concern about noise content or edge characteristics. More advanced techniques by using factors like noise content can also be used for image processing. According to Marr and Hildreth [7], such a filter need to have 2 features :

- The differential operator can calculate the approximation of the first or second order of derivative at every point in the image.
- The operator can be "tuned" to suite for any desired size. Large scale operator to detect blurry edges and small scale operator to detect sharply focused fine detail should be supported.

The Laplacian of Gaussian filter $\nabla^2 G$ satisfies the requirement. Here ∇^2 is the Laplacian operator $(\partial^2/\partial x^2 + \partial^2/\partial y^2)$, and G is the 2-D Gaussian function:

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Thus the Laplacian of Gaussian equation is expressed as:

$$\nabla^2 G(x, y) = [\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

With this expression the LoG filter of any size can be generated, such as the 5×5 filter showed in Figure-2.4.

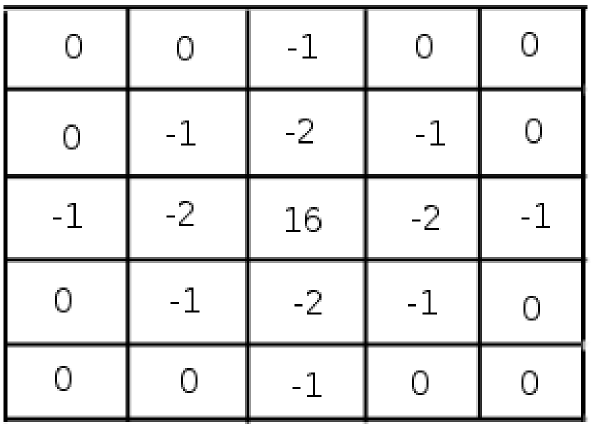
2.2.5 Canny Filter

Canny operator [3] is designed to find the "optimal" edges. An "optimal" edge means:

- Low error rate : all edges should be found
- Good localization : all edges should be as close as possible to the true edges.
- Single edge point response : the operator returns only 1 response for 1 true edge point.

The steps to apply Canny filter:

- Use Gaussian filter to smooth the image.



0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

(a) LoG Mask

Figure 2.4: LoG Filter Mask

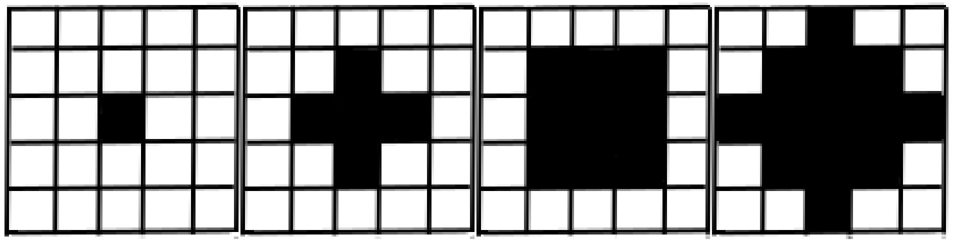
- Calculate the gradient magnitude and the angle images.
- Apply the nonmaxima suppression to the gradient magnitude image.
- Analyse and detect linked edges by using double thresholding and connectivity.

2.3 Segmentation: Morphological Watershed

Morphological Watershed [3] is a segmentation method to connect segmentation boundaries. This method treats a 2-D image together with its density as the third dimension parameter. Imagine that the image is cut into different segment according to the density of the pixel, then flood the image from the lowest value to the higher. Pixels with lower values will be quickly flooded. When the water rises to a certain level, two segment meet at some points, then a pixel on the deviding line will be found. As this process goes, the boundaries between segments could be obtained in the end. But this method is also not suitable for this case, because the original image (see Figure-1.1) has blurred background and the difference between droplet and background is not big enough. The processed result on the original image is shown in Section-3.1.7.

2.4 Morphological Operations

After the edge detection process another method is needed to extract the image components in the representation and description of region shape. The goal is to get the parameters of the droplet. Thus the morphological operations such as dilation, erosion, hole filling will be



(a) original **(b)** 4-connected dilation **(c)** 8-connected dilation **(d)** 8 diamond

Figure 2.5: Dilation

introduced in the next part.

2.4.1 Dilation and Erosion

The mathematical definition of dilation and erosion [3] is defined in the following:
Assume that A and B are sets in Z^2, dilation is denoted as $A \oplus B$, and erosion is denoted as $A \ominus B$,

$$A \oplus B = \{z|(\hat{B})_z \cap A \neq \emptyset\}$$

$$A \ominus B = \{z|(B)_z \subseteq A\}$$

The basic idea is doing spatial convolution operations with a morphological operator on the image. The dilation operation goes as described in the following : we have a 5×5 image with 1 pixel set in the center. The operator can be a 4-connected, 8-connected or diamond. When the operator moves to the center, the convolution operation is applied, and the neighbouring pixels will be set. In the example in Figure-2.5, the result of the 5×5 image after the dilation operation is identical to the operator.
The simplest applications of dilation is for bridging gaps. Suppose that two pixels are on the edge . The aim is to connect these 2 pixels. When the 4-connected operator is used, the result is shown in Figure-2.6.

Because the dilation operation adds a lot of extra pixels in the neighbourhood of the object, an operation which does just the opposite is needed. The erosion operation is just the function we want for this.
Suppose that an image is presented as in Figure-2.7a , and the erosion operator is a 4-connected operator or an 8-connected operator. After applying this operator the result is shown in Figure-2.7b and Figure-2.7c.

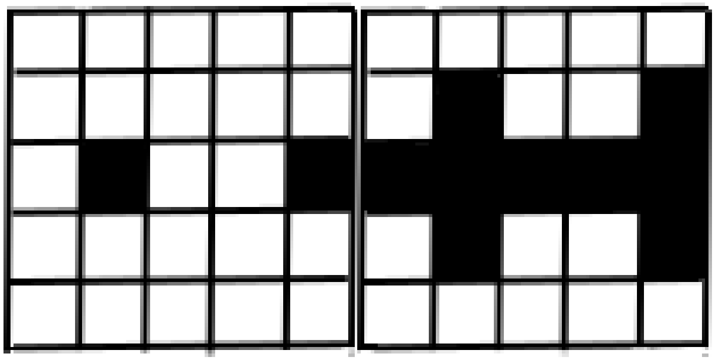

(a) isolated pixels **(b)** pixels connected

Figure 2.6: Pixels connected through dilation process

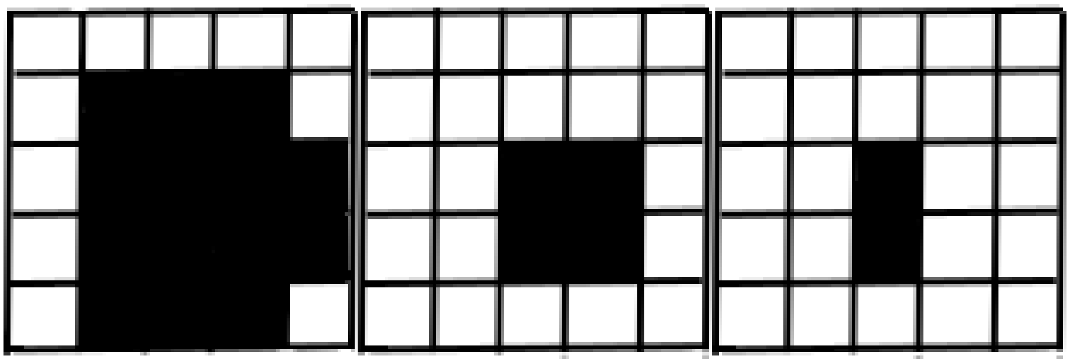

(a) original **(b)** 4-connected erosion **(c)** 8-connected erosion

Figure 2.7: Erosion

2.4.2 Hole Filling

One problem now exists : if isolated pixels on the edge need to be connected, the dilation function should be applied. But the extra pixels that the dilation operation adds are not needed. The erosion function does just the opposite thing of that the dilation function does, thus it can not be used directly to erase those extra pixels. The "dilation–erosion" process , which is also called "closing" [3] operation, can only be used for smoothing contours, eliminating small holes. These two methods are not enough to connect the isolated points on the long thin edges. So an extra function will be introduced , which is called "hole filling" [3]operation.
It contains 2 steps:

- check whether there are pixels surrounded by connected border pixels.
- if there are these type of pixels, assign some value(normally the same as the border pixels) to these pixels

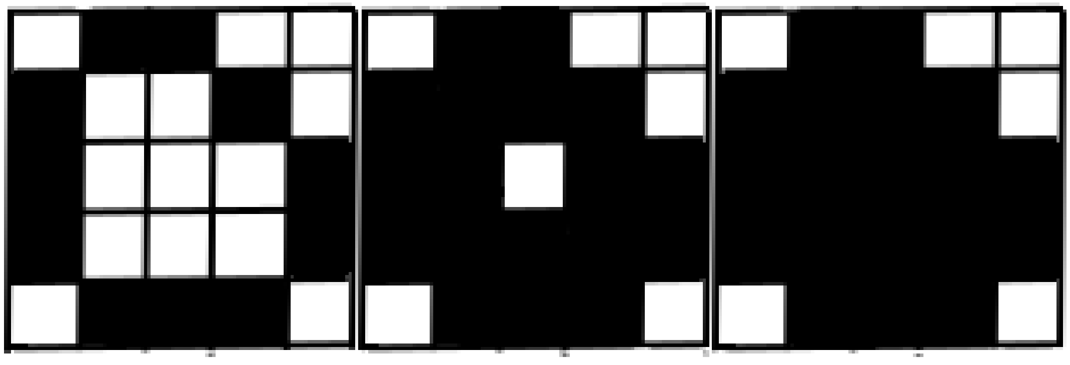

(a) original (b) first iteration (c) finish

Figure 2.8: Hole Filling

The hole filling operation can be implemented with the dilation operation: A loop with dilation operator on all the pixels within the border is processed. The process is shown in Figure-2.8.
After the hole filling operation, all of the pixels on the thin edge will not be erased by the erosion function. Thus the border of an object can be obtained through the "dilation–hole filling–erosion" process .

2.5 Additional Image Processing Methods

Doing modification on the input or output to get the best effect is one of the goals of this work. In this section some extra image pre-processing methods will be discussed. A lot of these functions can be found and used in Gimp [1].

2.5.1 Mean Filter

When an operator is used on an image, noise always comes out. Reason for the existence of noise is that the assumption of these filters is not always suitable unter all circumstance. For example, for the edge detection methods we assume that the pixel value varies quickly only on the edge. But in the real world this can also happen in the object or in the area of background. Thus noise pixels come out when the edge filter is applied . Most noise pixels are isolated, which means that by checking the values of the neighbourhood pixels these pixels could be identified, then with a threshold it is possible to erase these noise pixels.
The mean filter [3] is used for smoothing an image. A normal version of this filter is

$$x = (\sum_{i=1}^{9} Z_i)/9$$

If the mean filter is applied on the original image , the job to extract the border will be harder

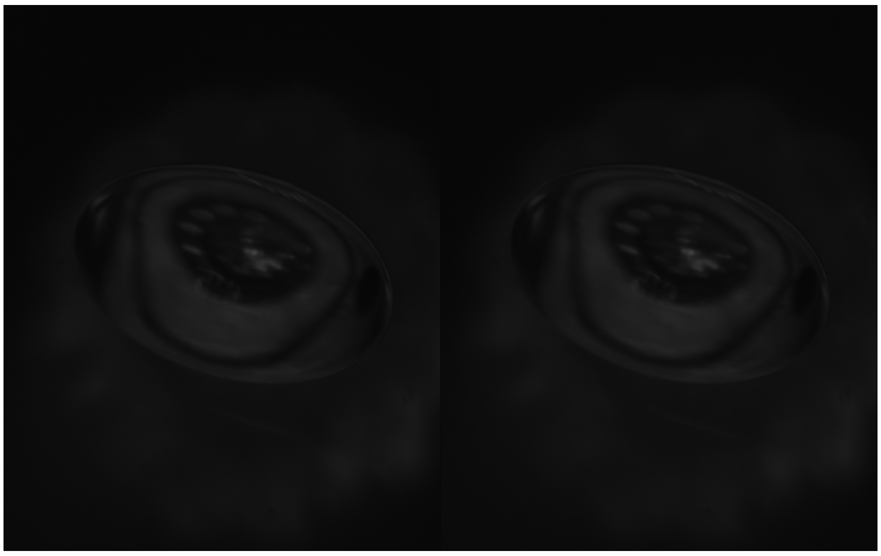

(a) original image (b) applied with mean filter

Figure 2.9: Mean Filter

. Because this method makes the difference between object and background smaller.
The boundaries on the original droplet image are already not clear enough. The difference between the 2 images can not be identified by eye easily(see Figure-2.9). But actually the result of edge detection operation on the smoothed image will be worse than that on the original one .

2.5.2 Antialias Filter

Antialias [1] is a process to reverse alias, or to make the jagged edge smoother. It adjusts the values of pixels on the boundary. After this operation, the changes of the pixel values will not be so abrupt.
An example is shown in Figure-2.10.

2.5.3 Deinterlace Filter

When a moving object need to be recorded , especially when this object moves very fast, the obtained image looks always blurred and stripped. This problem comes with the working mechanism of camera. Normal cameras record the double amount of images per second, but all of them have only half vertical resolution. After the recording step every 2 images will be interlaced into 1 image by interlacing them in the following way: first line of first image, then first line of the second image, then the second line of the first image , then the second line

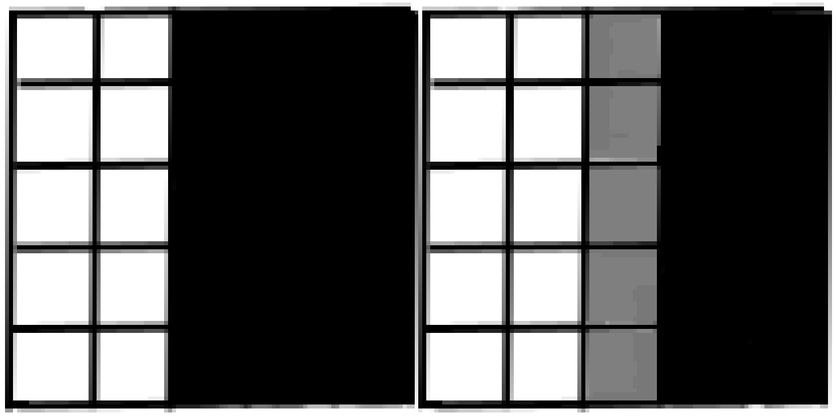

(a) original image (b) filtered image

Figure 2.10: Antialias Filter

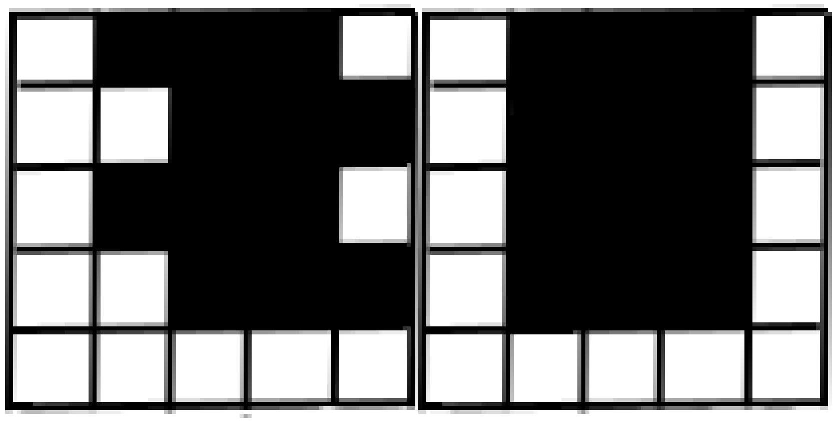

(a) original image (b) filtered image

Figure 2.11: Deinterlace Filter

of the second image , and so on. When the object moves very fast, it is possible that the 2 images will no longer be compatible with each other.
The following is a simple approaches that can realize the Deinterlace Filter [1]:

- Calculate the possible offset of the second to the first image.
- Choose lines to keep. For example: keep the even lines.
- According to this offset adjust all the pixels in the odd lines or vice versa.

An example is shown in Figure-2.11.

2.5.4 Despeckle Filter

Despeckle Filter [1] is used to remove small defect caused by dust or scratches. The basic idea is replacing the defective pixels with some valid values.
Here is a simple approach to realize this filter:

- Select the defect areas. This is normally done by hand.
- Set a proper radius. It is better to set the radius bigger than the radius of the defect areas.
- Calculate the median value of the pixels within the specified radius.
- Replace the defect pixels with the median value.

This filter is similar to the mean filter, both of them can be used to remove noise.
Advantages of this filter over the mean filter:

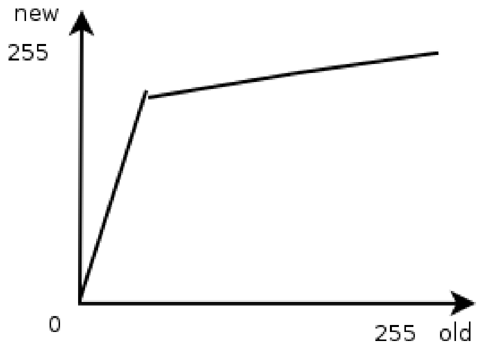
- Easy to implement.
- Sorting is much faster than multiplication and division.

Disadvantages of despeckle filter over the mean filter:

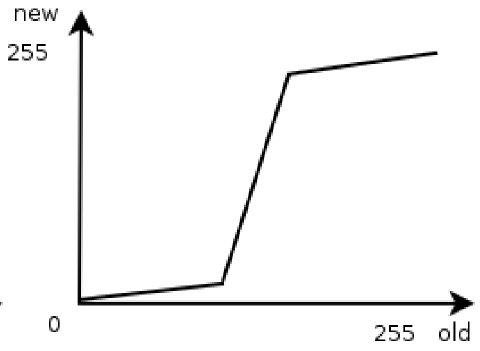
- The defect area must be specified.
- The radius must be specified. It can not be set beforehand, otherwise when the radius is too small, this filter will not work.
- The median is not always a proper value.

2.5.5 Contrast Enhancement Filter

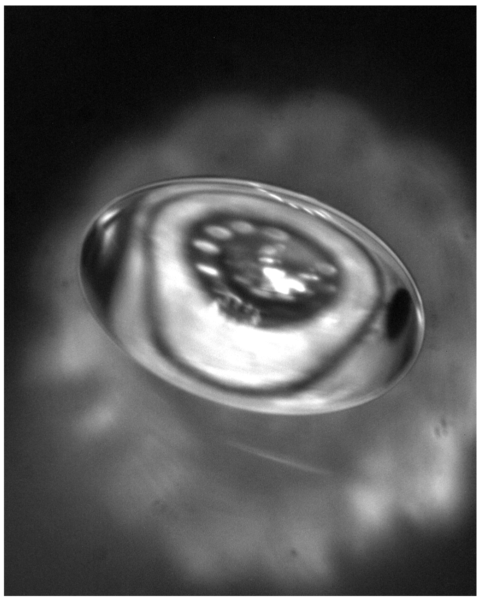
This filter maps the values of pixels in the original image to a range so that the image can be identified much easier.
A simple example is shown in Figure-2.12. The process can also be done in Gimp.[1] The result of applying an adjust contrast enhancement filter onto the original droplet image is also shown here. Although the result of the droplet image seems good, this filter is still not suitable for the edge detection methods applied on our droplet image.
The biggest problem is this filter doesn't change the relationship between adjacent pixels. The pixel values of the droplet are not significantly different to those of the background. Edges can only be detected by calculating the difference between pixels. During the boundary detecting process, the difference between the adjacent pixels are calculated. Then a threshold is also calculated. According to this threshold these differences should be evaluated . Once they fulfill the requirement of the threshold, the related pixels become the potential edge pixels.
Two aspects are important in the process: one is the difference between two pixels , the other one is the threshold. The absolute value of the difference ifself is not so important, because when the difference is small, the threshold can also be reduced to fit this situation. This



(a) filter example 1



(b) filter example 2



(c) filtered image

Figure 2.12: Contrast Enhancement Filter

produces more noise .

3 Generic Composited Method

In this chapter the method to extract the droplet object from the image and ways to calculate the properties of this object will be discussed.

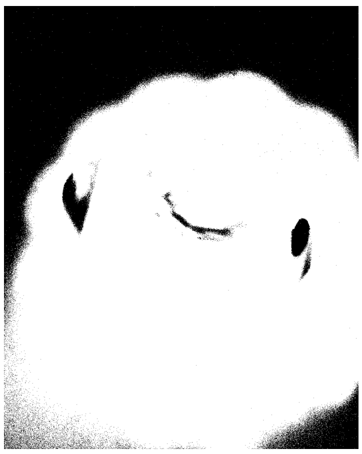
3.1 Comparison of Object Extracting Methods

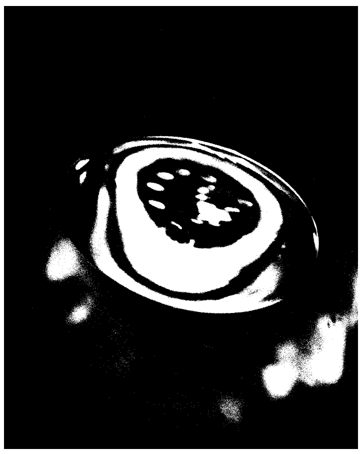
In Chapter-2 several edge detection methods to extract object boundaries were introduced. The possible methods that can be applied on the droplet image are:

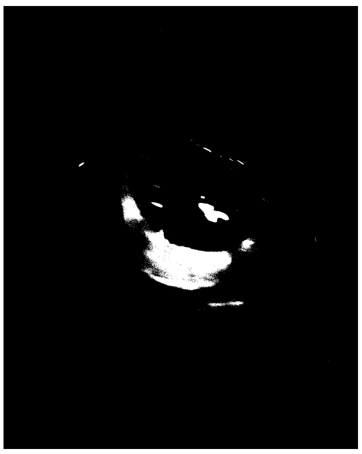
- simple threshold in Section-2.1
- line filters in Section-2.2.1
- prewitt filter in Section-2.2.2
- sobel filter in Section-2.2.3
- Laplacian of Gaussian filter in Section-2.2.4
- Canny filter in Section-2.2.5
- morphological watershed method in Section-2.3

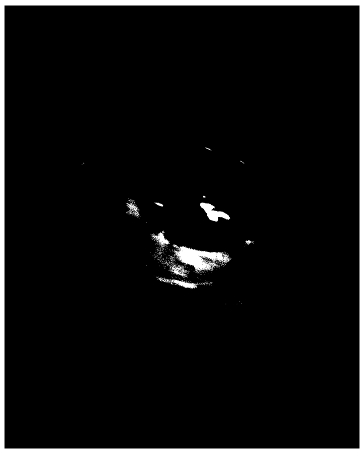
3.1.1 Simple Threshold

In our case it is not possible to apply the simple threshold on the droplet image, because it's hard to distinguish the value range of the droplet from the value range of the background. Most of the pixels in the original droplet image have values between $0 \rightarrow 30$. In order to see the effect of the threshold filter, the value 10, 20, 30, and the global threshold value 33.5 and the half of the global threshold value 16.75 were applied. The results are as shown in Figure-3.1.
With the threshold 20 a general shape of the droplet has been obtained. The problem is that part of the boundaries was missing.
With threshold 10, 30 or 33.5 nothing could be obtained.
With threshold 16.75 (half of 33.5) a droplet object with a lot of noise below the droplet was obtained.
Thus the simple threshold method is not suitable in our case.


(a) threshold: 10


(b) threshold: 20


(c) threshold: 30


(d) global threshold: 33.5

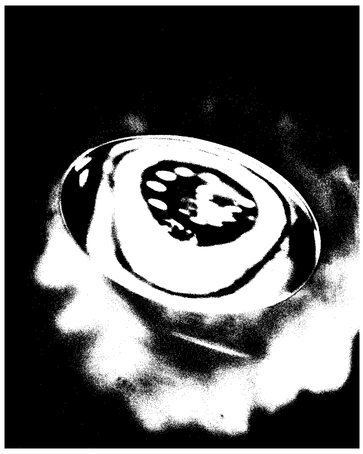
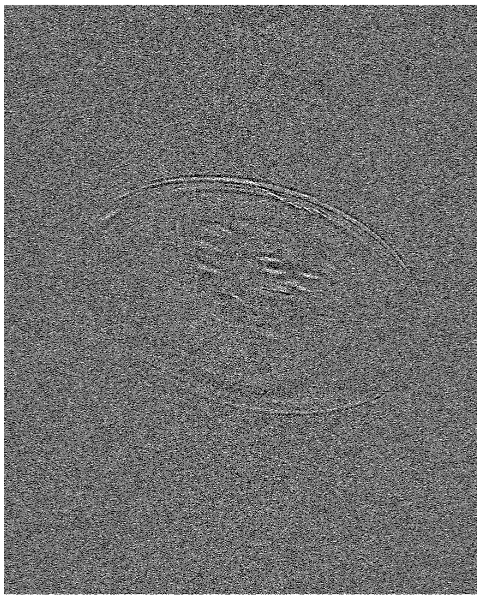
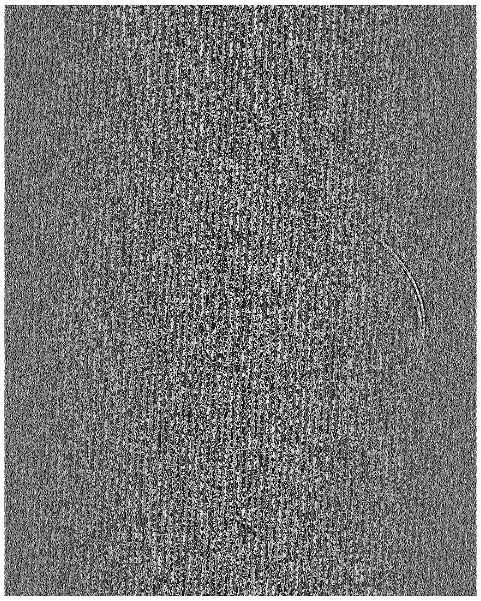

(e) threshold: 16.75

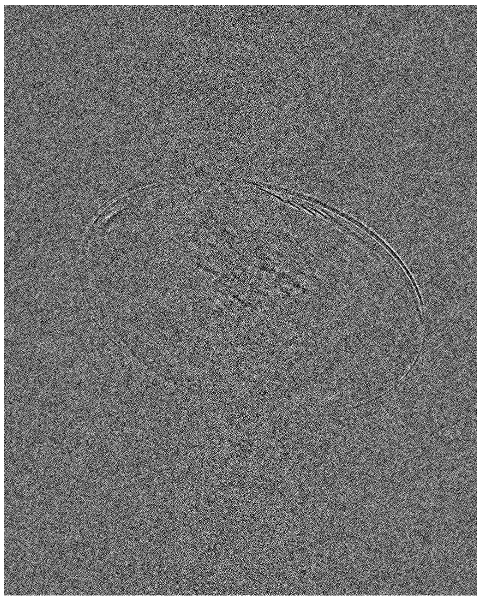
Figure 3.1: Simple Threshold Filter Results

3.1.2 Line Filters

Actually this kind of filters (vertical, horizontal and diagonal) generated nothing but a lot of noise , which are presented in Figure-3.2. None of these filters worked on the original image. The results indicate that the line filters are also not suitable here. The boundary of the droplet can not be seen.


(a) horizontal filter


(b) vertical filter


(c) diagonal filter 1

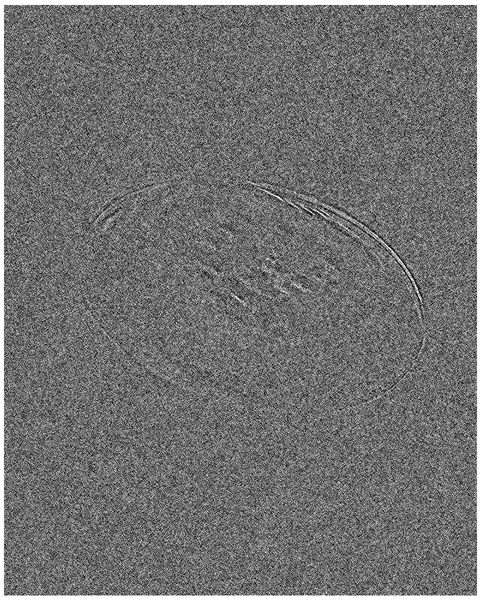

(d) diagonal filter 2

Figure 3.2: Line Filter results

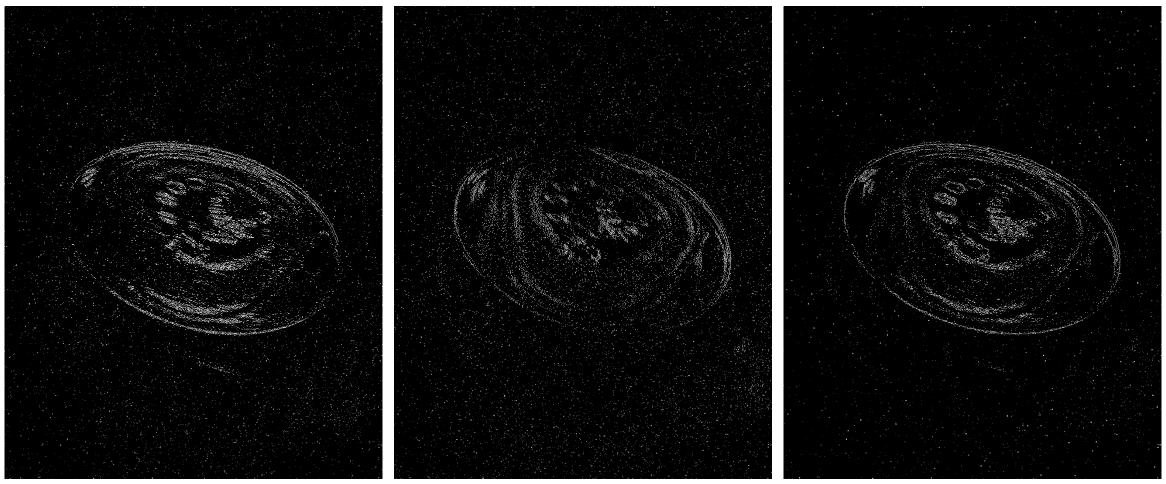

(a) in horizontal direction (b) in vertical direction (c) in both direction

Figure 3.3: Prewitt Filter Results

3.1.3 Prewitt Filter

Applied with the prewitt filter (horizontal, vertical and both directions) on the original image , the results as shown in Figure-3.3 have been obtained .

The results showed most part of the boundary clearly. The horizontal filter missed a lot of edge pixels on the left and right side of the droplet. The vertical filter missed a lot of the edge pixels on top and bottom side of the droplet. A good shape of the droplet was obtained when the results of both directions were combined, although some small part of the boundary was still missing. The image applied with prewitt filter (with both directions) had a better outcome. Only a small part on the left-bottom side of the droplet was missing. The amount of noise in the both direction filtered image was relatively less than that in the images filtered with only one direction.

Problem of this filter was that most generated pixels were on the border. and a lot of them were not connected. So an additional method should be introduced here for further operation.

3.1.4 Sobel Filter

Sobel filter performs similarly as prewitt filter. The sobel filter (in horizontal, vertical and both directions) was applied on the original image , the results is shown in Figure-3.4.

The images with horizontal or vertical filter have the same problem just like that with prewitt filter: when the horizontal filter has been applied, part of the boundary on the left and the right side were missing. When the vertical filter has been applied , part of the boundary on top or at the bottom of the droplet were not presented.

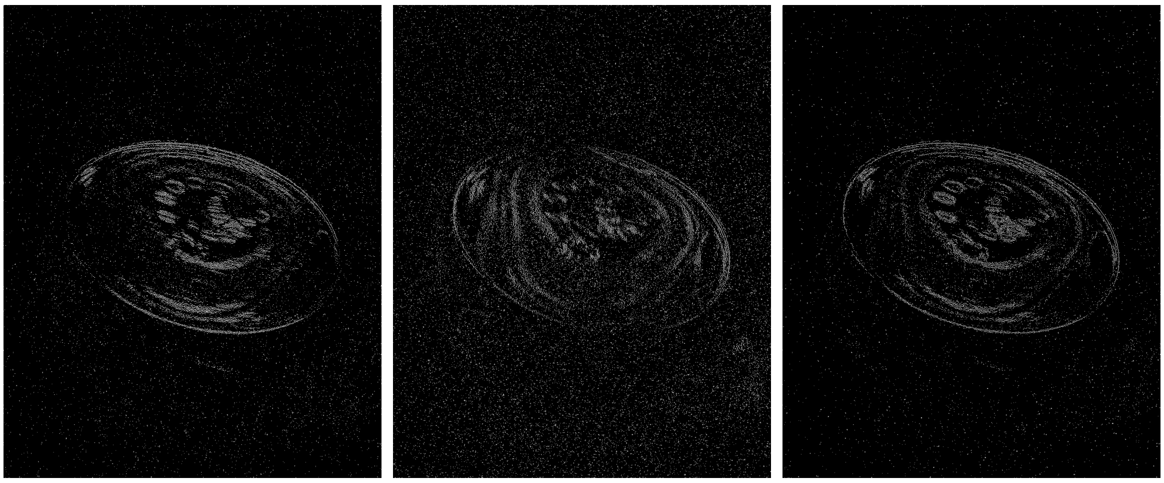

(a) in horizontal direction (b) in vertical direction (c) in both direction

Figure 3.4: Sobel Filter Results

But the output with filter in both directions looks like to be perfect. Only a very small part on the left-bottom side was missing.
The problem with this filter is the same with prewitt filter: although the generated pixels were on the boundaries, a lot of them were isolated. An additional method should also be used to achieve a better outcome.

3.1.5 Laplacian of Gaussian Filter

The Laplacian of Gaussian filter was applied on the original image. The result image (Figure-3.5) shows that the LoG filter can certainly extract the droplet shape from the image. But the noise could not be handled by LoG filter .
Most of the result image was covered with useless noise pixels , a lot of them had relative big size so that they could not be easily removed by simple noise removing methods.
Thus this method is also not suitable .

3.1.6 Canny Filter

Canny filter has the same issue as LoG filter, although it is not so severe as that with LoG filter. Figure-3.6 shows the result of applying the Canny filter on the droplet image. The boundary of the droplet could be seen, but the noise has taken almost half of the area which this filter has extracted. Canny filter is also not suitable .

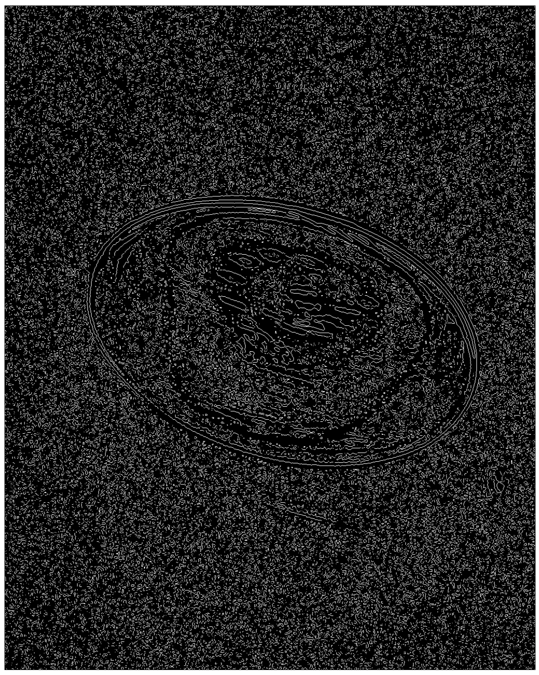

(a) filtered image

Figure 3.5: LoG Filter

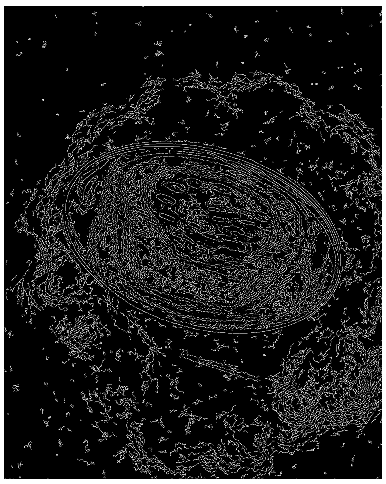


Figure 3.6: Canny Filter

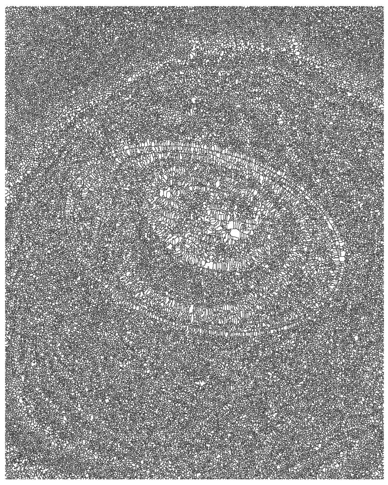


Figure 3.7: Morphological Watershed

3.1.7 Morphological Watershed Filter

The result after applying the morphological watershed filter is shown in Figure-3.7. The outcome was very bad. The border of the droplet could not be seperated from the noise. The morphological watershed filter could not simply be applied on the original image.

3.1.8 Comparison and Conclusion

The results of the filters showed that most of them can not process the image with a satisfying result, only because the original droplet image has the following features:

- The value range of the droplet and that of the background have overlapping part.
- The value varies on the whole image, not only on the boundary of the droplet. Thus some filters that are sentitive to the value changes have generated a lot of noise.

To extract the droplet object from the noise is not easy, so the Canny filter was not chosen. The line filters , the LoG filter , the morphological watershed filter have failed to generate a identifiable droplet shape, thus they were not be chosen. The threshold method could get most part of a droplet shape, but parts of the droplet were still missing.
Only prewitt filter and sobel filter have generated relative satisfying results: the results were with relatively clear boundaries and less noise. But some additional methods need to be added to achieve our aim.

Filter name	Results
simple threshold	The result depends on the value of the threshold. Not easy to get the whole object of the droplet.
line filters	Only generated noise.
prewitt filter	The result of using the both directions filter looks good. Need extra function to connect the isolated pixels on the edge.
sobel filter	similar to prewitt filter.
LoG filter	The shape of the droplet was obtained, but a lot of noise was also generated.
Canny filter	similar to LoG filter, with less noise, but still not practical for the original droplet image.
morphological watershed	It generated only noise

Table 3.1: Comparison of Image Extracting Methods

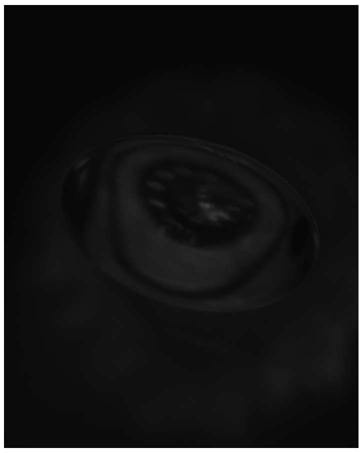
3.2 Comparison of Pre-Processing Methods

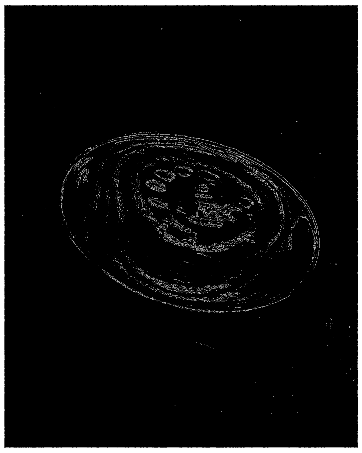
In Section-2.5 several methods for image pre-processing were introduced. All of them were applied on the original droplet image and was checked whether they could improve the performance of the object extracting methods .
The introduced 5 methods are :

- Mean filter
- Antialias filter
- Deinterlace filter
- Despeckle filter
- Contrast Enhancement filter

3.2.1 Additional Pre-Processing Methods

In this part the pre-processing methods on the original image were firstly applied, after that the object extracting methods on the processed images were used. Then the morphological operations were applied to see whether the outcome could be better or not.
According to Section-2.5, prewitt filter and sobel filter were applied on the processed image.


(a) first with Mean Filter


(b) then with Prewitt Filter

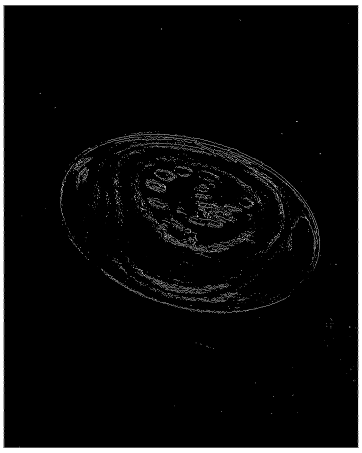
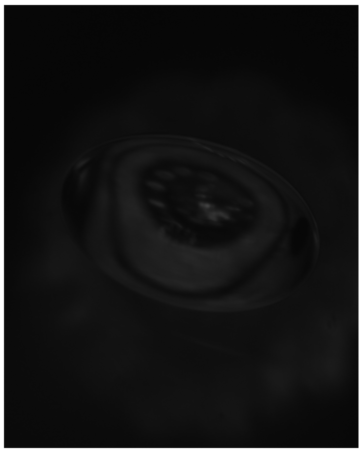
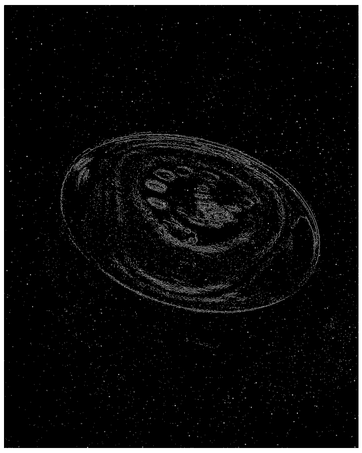

(c) or with Sobel Filter

Figure 3.8: Pre-Processing with Mean Filter


(a) with Antialias Filter


(b) then with Prewitt Filter

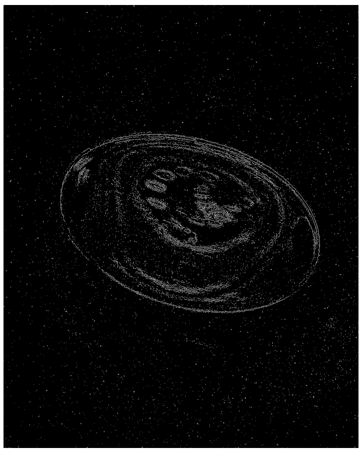
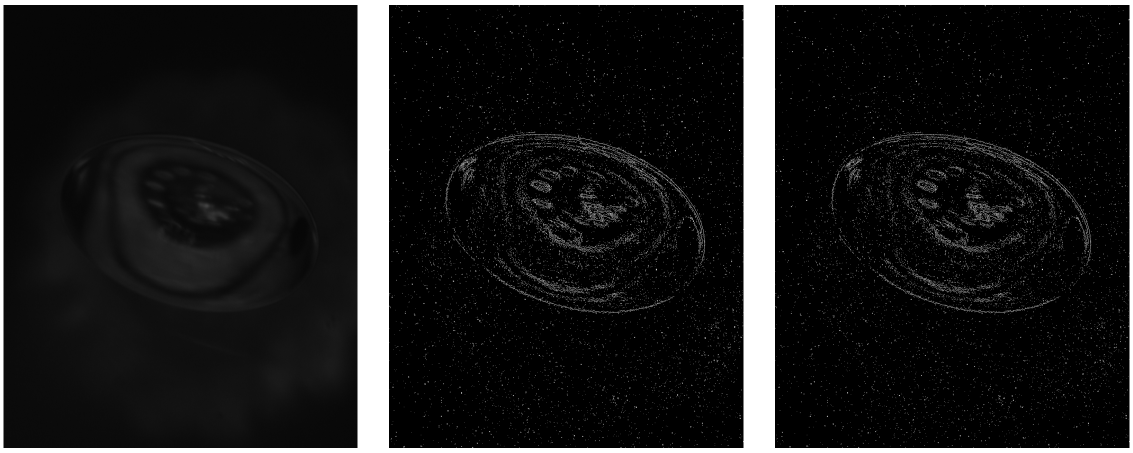

(c) or with Sobel Filter

Figure 3.9: Pre-Processing with Antialias Filter

3.2.1.1 For Edge Detection Methods

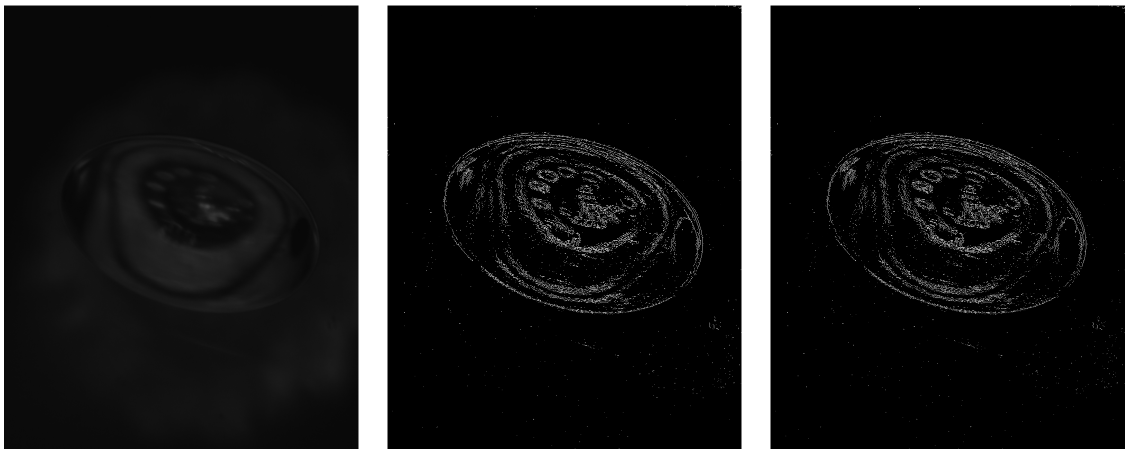
The first one was doing object extraction with the 3×3 mean filter filtered image: The mean filter was firstly applied, then the output was processed by prewitt filter or sobel filter. The results are shown in Figure-3.8.

The next try was with the antialias filter : The antialias filter was applied on the original image, after that prewitt filter or sobel filter was used. The results are shown in Figure-3.9. And the next one was the results with the deinterlace filter: The result applied with the



(a) with Deinterlace Filter **(b)** then with Prewitt Filter **(c)** or with Sobel Filter

Figure 3.10: Pre-Processing with Deinterlace Filter



(a) with Despeckle Filter **(b)** then with Prewitt Filter **(c)** or with Sobel Filter

Figure 3.11: Pre-Processing with Despeckle Filter

deinterlace filter is shown in Figure-3.10a, after that prewitt filter or sobel filter was used. The results are shown in Figure-3.10b and Figure-3.10c.
And the next one was with the Despeckle filter. The procedure went as antialias filter or deinterlace filter : first with the despeckle filter, then with prewitt filter or sobel filter. The results are shown in Figure-3.11.

The last part was doing the process with the contrast enhancement filter. The results are shown in Figure-3.12.

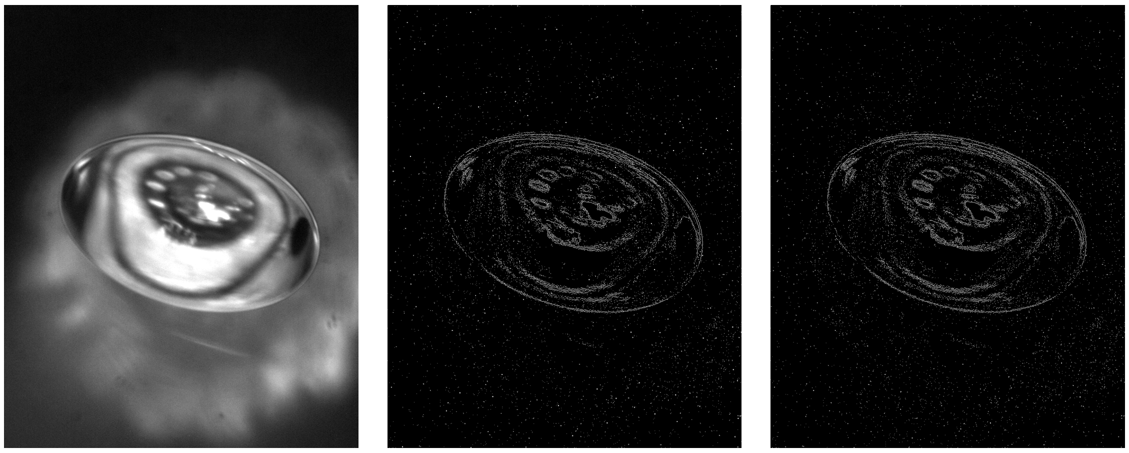

(a) contrast Enhancement (b) then with Prewitt Filter (c) or with Sobel Filter

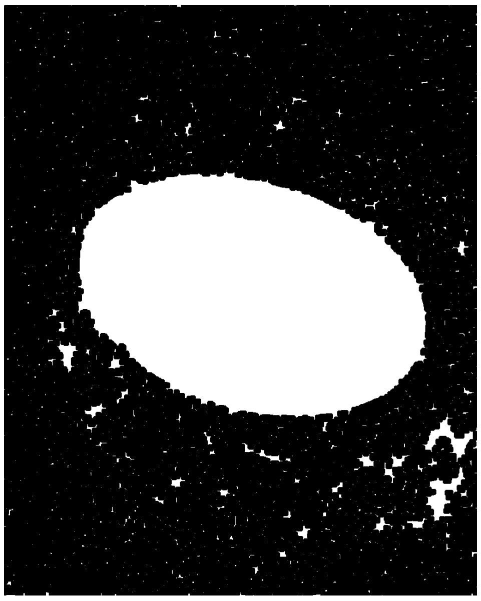
Figure 3.12: Pre-Processing with Contrast Enhancement Filter

3.2.1.2 Comparison by using Morphological Operations

In this part the morphological operations were applied on the input that were described in Section-3.2.1.1, and the results were compared. The one with better outcome was chosen. At first the original image was directly applied with prewitt filter or sobel filter. No other pre-processing methods were applied . The results are shown in Figure-3.13.

The next one was the image filtered by the Mean filter . The results are shown in Figure-3.14. The results indicated that the filtered images were not suitable to apply the morphological operations. The gap between pixels on the edge was too big to fill. Only two whole white pictures were obtained after the morphological operation, that means that this operation has totally failed .

The next one was images filtered by the antialias filter . The results are shown in Figure-3.15. The next part was to apply the morphological operations on the images filtered by the Deinterlace filter. The results are shown in Figure-3.16. The filtered image has generated a lot of extra noise pixels. It was not suitable for our case. The next one was the image filtered by the Despeckle filter. The results are shown in Figure-3.17. The outcome seemed to be perfect. In the statistical comparison Table-3.2 , this one was also the best.
But the problem with the Despeckle Filter was as mentioned in Section-2.5.4: implementation of this filter without human intervention is very difficult. And the last one was with the contrast enhanced images. The results are shown in Figure-3.18. The results also indicated that this method was also not suitable for further processing. Conclusion on the results : in the first step the antialias filter could be used, or none of these methods should be applied. After that the morphological operation could be used. Thus a relatively good result could be obtained.


(a) with Prewitt Filter

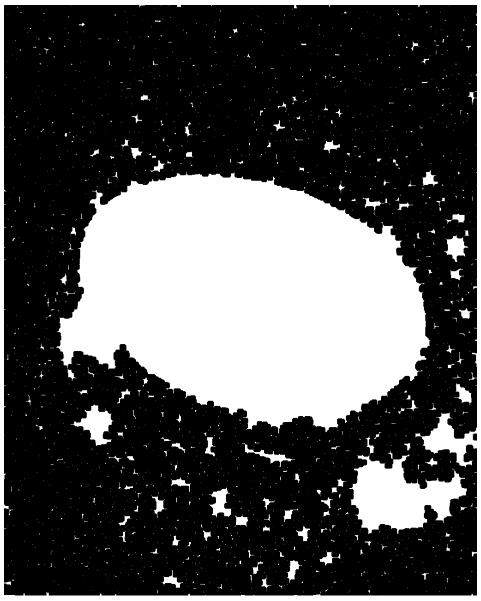
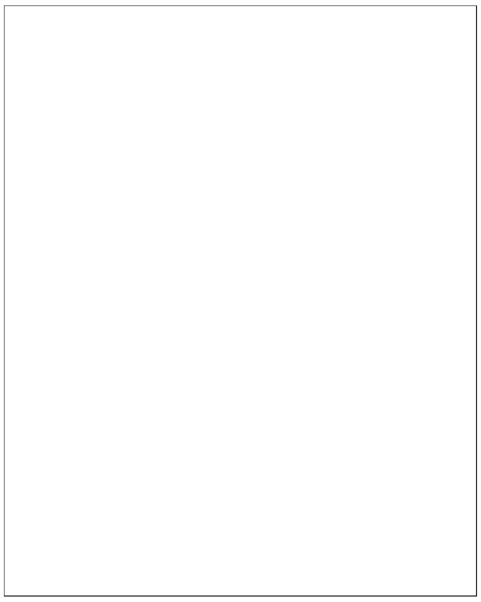

(b) with Sobel Filter

Figure 3.13: Morphological Operation Results on Original Image


(a) with Prewitt Filter: failed, all pixels were white

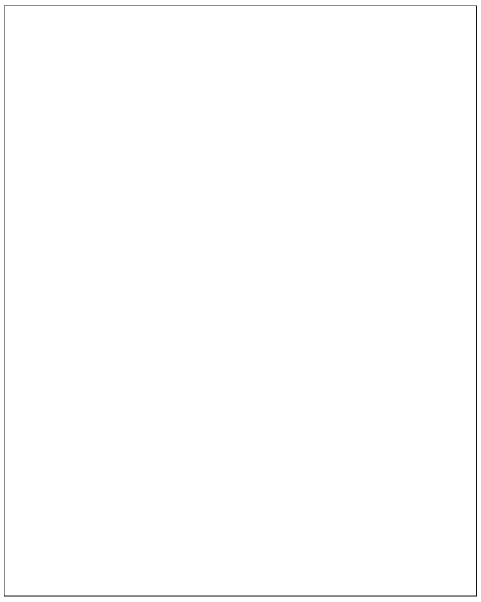

(b) with Sobel Filter: failed, all pixels were white

Figure 3.14: Morphological Operation Results on Mean Filter filtered Images

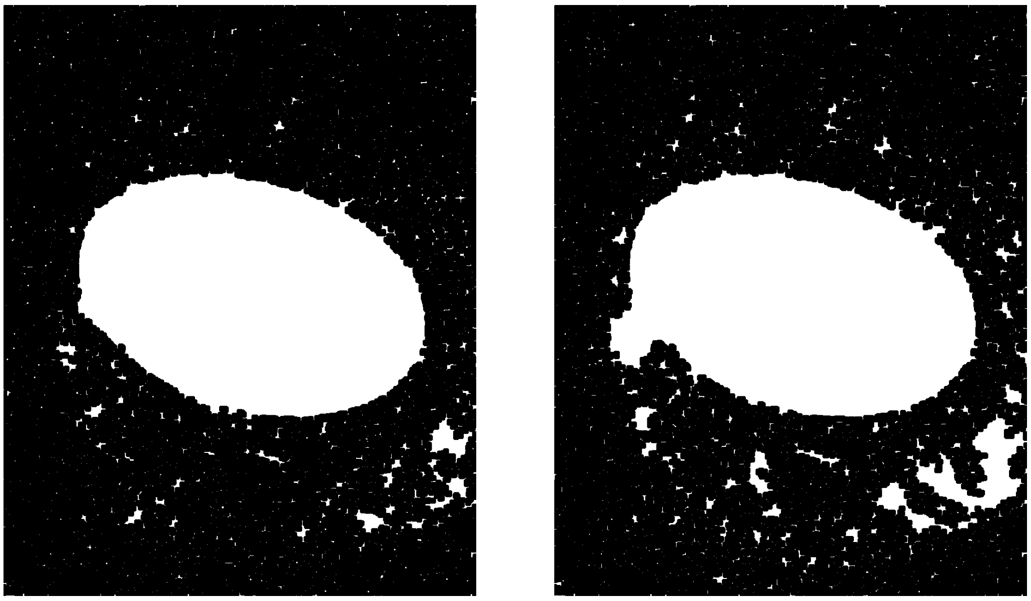

(a) with Prewitt Filter **(b)** with Sobel Filter

Figure 3.15: Morphological Operation Results on Antialias Filter filtered Images

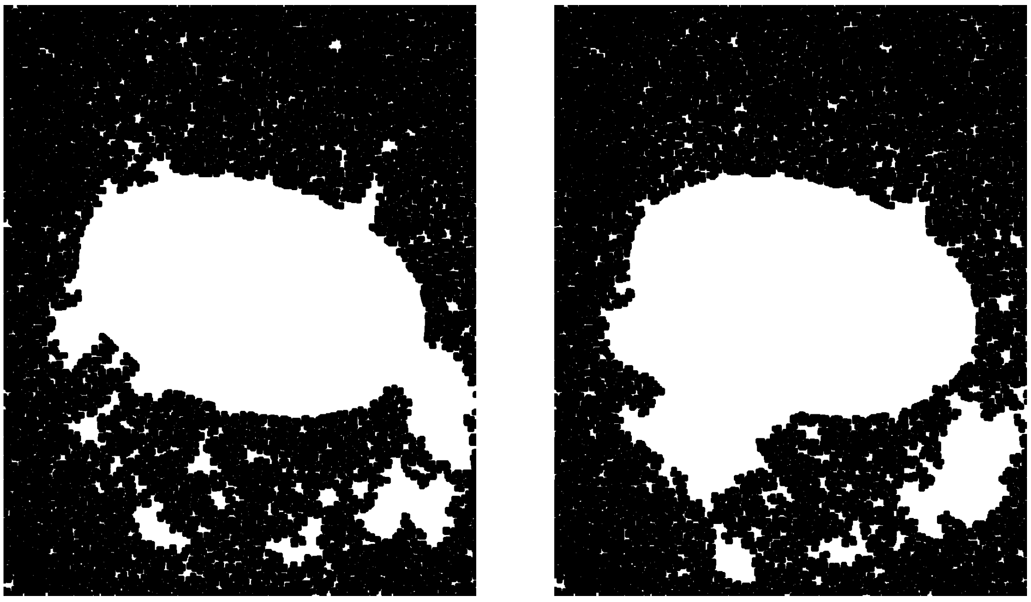
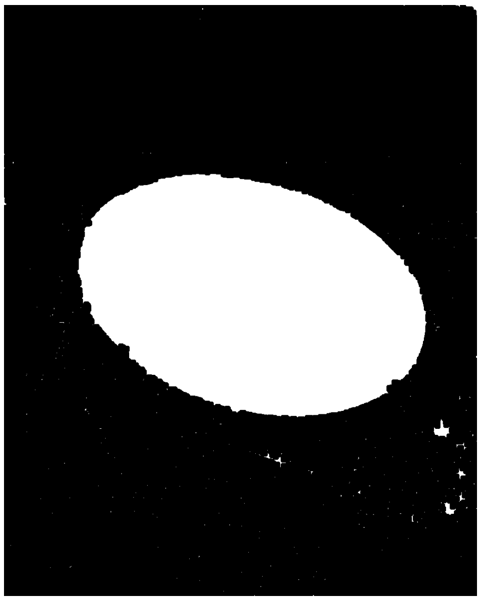

(a) with Prewitt Filter **(b)** with Sobel Filter

Figure 3.16: Morphological Operation Results on Deinterlace Filter filtered Images


(a) with Prewitt Filter

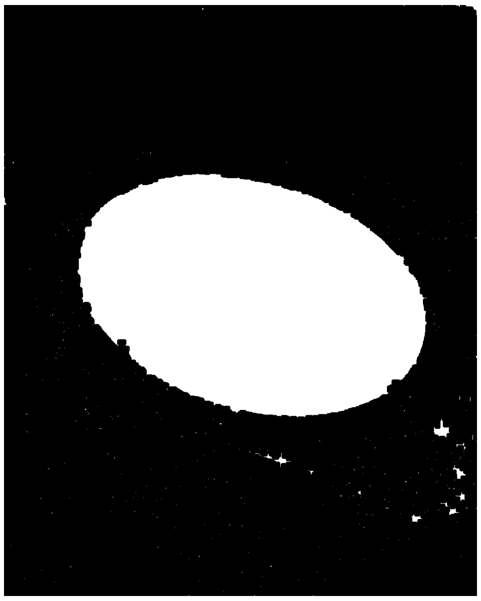
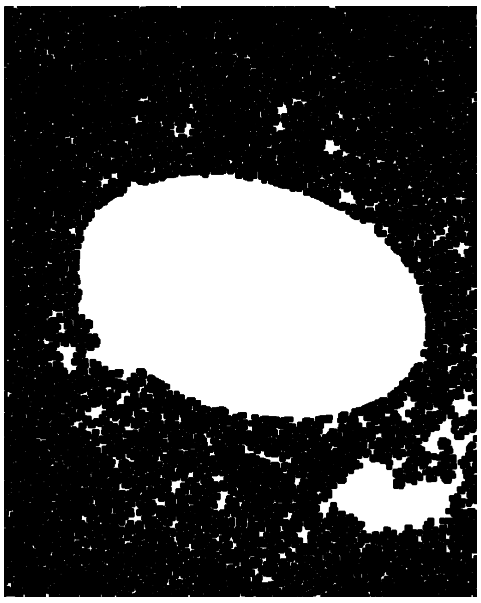

(b) with Sobel Filter

Figure 3.17: Morphological Operation Results on Despeckle Filter filtered Images


(a) with Prewitt Filter

(b) with Sobel Filter

Figure 3.18: Morphological Operation Results on Contrast Enhanced Images

3.3 The Matlab Implementation and Geometry Calculation

In this section some useful functions in matlab are introduced.

3.3.1 Image Processing Methods in Matlab

There are built-in functions for edge detection [4], for example :

```
[e_img_rob , threshold_e_rob] = edge (Orig, ’roberts’ );
[e_img_pre_v , threshold_e_pre_v] = edge (Orig, ’prewitt’,  ’vertical’);
```

The common edge detection methods such as prewitt method, sobel method, Canny method, Laplacian of Gaussian method are implemented, the correspondent parameters for these methods are : prewitt, sobel, canny, log, and for prewitt and sobel filter the connectivity parameter can also be given by using keywords like vertical, horizontal or both . This function feeds back a filtered matrix and the threshold used for the matrix. The threshold is usually calculated by using the standard distribution method.
For the morphological operations, 3 functions in Matlab can be found to fulfill our requirements: imdilate, imfill, imerode. Example for that is just as follows:

```
img_tmp = imdilate(img_sob_b, strel(’disk’, 7));
img_tmp = imfill(img_tmp, ’holes’);
img_tmp = imerode(img_tmp , strel(’disk’, 7));
```

The need parameters that need to be specified are :on which image or matrix the function will be applied , how big the radius of the affected area is. Matlab also provides method to calculate the global threshold . This function is called “graythresh(m)”, m is the input matrix.

3.3.2 Geometry Calculation in Matlab

Matlab provides a built-in function , which is called “regionprops” , to measure the geometric properties of an image.
The basic usage of this function is as shown:

```
STATS = regionprops(Image, properties)
```

"properties" in this function has a lot of values, such as "Area", "Centroid", "MajorAxisLength" .etc . Calculation of the area , the length of the major- and minor-axis of the droplet is the goal, thus "FilledArea", "MajorAxisLength", "MinorAxisLength" should be the minimun set of the properties parameters.
Although the filtered images worked with the regionprops function, presenting the droplet itself was preferred. Thus all the other noise should be eliminated.
Matlab has no built-in function for that, but another way could be used: keep the object with the biggest area , or the longest MajorAxisLength, then erase all the other objects.
This method was implemented as follows:

- Calculate the "FilledArea", "MajorAxisLength", "MinorAxisLength", "PixelList" properties with regionprops function
- Use "max" function to find the ID of the biggest object.

Another feature of this procedure was : the ID number of the droplet can also more or less reflect how many noise clusters were in the processed image. But it didn't indicate how many noise pixels exist.

3.3.3 The Algorithm

Now the droplet could be extracted and features of the droplet could be calculated. The algorithm could be described as follows:

- smooth the image, or do nothing.
- detect edges.
- do morphological operations to connect the isolated pixels on edges.
- extract features of the droplet.

3.3.4 Comparison of Geometry Calculation Results

When the droplet has been extracted from the image , the properties calculation should be done on the droplet. The "regionprops" function was applied on the images from Section 3.2.1.2, The properties of the droplet such as area, axislengths of the object were obtained. The result is shown in table-3.2. The despeckle method could generate the most clean result as discussed in Section-3.2.1.2. Although it was not chosen as part of our algorithm, the result on the despeckle filtered image could be used as a standard to check whether a method is good.
The comparison indicated that the image without the additional pre-processing methods worked well.
The prewitt method and sobel method both worked fine.

Pre-Processing Method	Edge Detection Method	Major Axis Length	Minor Axis Length	Area
No	Prewitt	776.9	495.3	301910
	Sobel	803.6	513.6	319301
Mean	Prewitt	1478	1182	1310720
	Sobel	1478	1182	1310720
Antialias	Prewitt	776.9	497.6	303172
	Sobel	798.6	506.3	312956
Deinterlace	Prewitt	937.6	523.9	355126
	Sobel	815.8	634.7	381056
Despeckle	Prewitt	773.2	492.0	298296
	Sobel	772.4	491.8	298161
Contrast	Prewitt	783.4	505.8	309620
	Sobel	1040.8	563.5	392874

Table 3.2: Comparison of Geometry Calculation Results

In the next chapter sobel filter was chosen as the edge detector to realize the single-pass algorithm.

4 Single-Pass Implementation

In Chapter-3 we have compared the image processing methods and introduced an appliable image property calculating method. The final processing procedure consists of the following steps:

- 1. Image smoothing.
- 2. Edge detection.
- 3. Image Dilation.
- 4. Image Hole Filling.
- 5. Image Erosion.
- 6. Features extracting function.

The morphological operations should be seperated into different steps only because each operation in this list generates a new image. In some circumstances this requires too many memory space. And when this algorithm is applied, waiting for the arrival of the entire image data is necessary. This wastes a lot of time.

4.1 Memory Space Usage Analysis

The needed memory space can be calculated as follows:
The original image file have a size of 1280×1024 . Assume that for each pixel a single unit of memory is needed to save the value. Thus $1280 \times 1024 \times 1 = 1310720$ units are needed to save the temporary data.
For each step in step 1, step 2 , step 3, step 4, step 5 a sample of the temporary data is needed, thus $1280 \times 1024 \times 1 \times 5 = 6553600 = 6M$ units of memory space are needed .
In this procedure, the algorithm stops 5 times to wait for the arrival of the whole temporary data of the image, which is shown in Figure-4.1. Thus the optimization of the algorithm is needed.

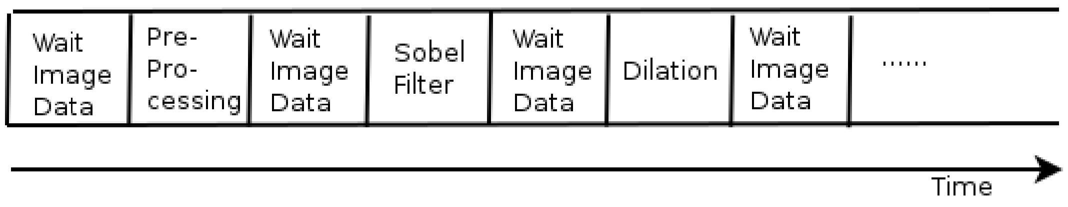


Figure 4.1: Process Delay

4.2 Algorithm Analysis

In this section the image processing methods were looked into to see whether they could be improved.
If there is no possilibity to improve the method, a comparable method would be introduced and then a relatively good result should come out.
Sobel method was used in the following section for discussion.

4.2.1 Sobel Method

The content of sobel method in Section-2.2.3 showed that the sobel method convolves the original image with sobel filter. In our case a 3×3 sobel filter was used.
Although the Matlab built-in function applied this method with a whole image as its input, the real data that was operated on was just in a 3×3 matrix. and the outcome was a value of the centering pixel.
Then the sobel method could be modified into :

- Read the data of a 3×3 matrix from the original image.
- Apply the sobel method.
- Return the value of the centering pixel.

It was difficult to read 3 pixels in the first row, then jump to the second row , and to the third row. When the second pixel was processed , jumping back to the first row to read the required data was needed.
Thus a temporary memory was also needed . In our case 2 rows and 3 extra pixels would be enough. The conclusion: To implement the sobel method 2 rows and 3 pixels of data should be read, which was $2 \times 1024 + 3 = 2051$ units of memory. The waiting time for image reading was shortened from 1280 rows to 2 rows, after that the reading operation was done parallelly with the image processing procedure.
Figure-4.2 indicates the situation. In this graphic, "Z_i" means the area where the sobel filter can be applied, "D" means the already read data, "U" means the still not read data.

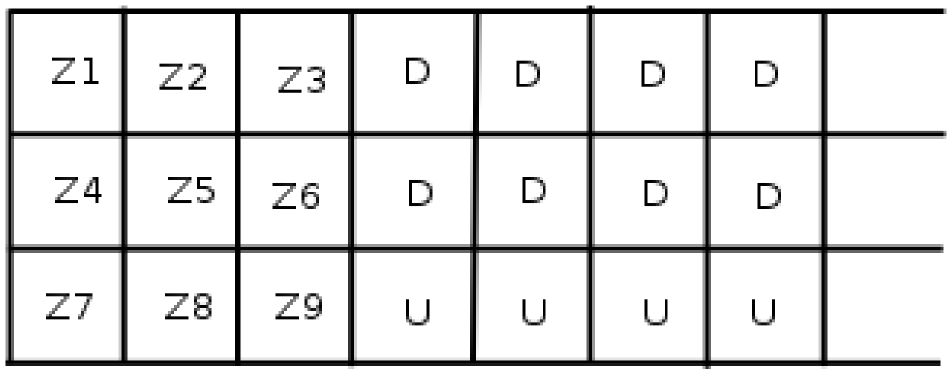


Figure 4.2: Sobel Method Memory Usage

4.2.2 Dilation Method

The dilation algorithm worked similarly as the sobel method. Just as discussed in Section-2.4.1, and in Section-3.2.1.2 the dilation and erosion filter with the size of 7×7 were used. To fulfill the requirement of the filter 6 rows and 7 pixels should be read at first , then the dilation method could be used.
The output was the value of the centered pixel.
The total memory usage was $6 \times 1024 \times 1 + 7 = 6151$ units.
The waiting time was shortened from waiting 1280 lines to waiting 6 lines .
For the implementation of dilation method the whole 7×7 area should be scanned , when a pixel which has value bigger than the “blank value” (In our case we use 0) was encountered, the output was set to 1 or 255. The optimal situation is that in each scanning process the first pixel already has a bigger value, then the next 48 pixels will not be looked into. In this way the processing time would be reduced.
Also the worst case exists: all pixels in the 7×7 matrix contain only “blank value”. Thus all 49 pixels should be scanned.

4.2.3 Hole Filling Method

This part is about one of the two difficult parts of the algorithm modification. Reason for the difficulty is that the hole filling method can only be applied on the whole image.
In Section-2.4.2 a method was introduced: At first the pixel set of connected boundaries is obtained. After that all pixels within this boundaries could be assigned with some values (normally the value of the border pixels). The second step could be applied with the recursive dilation function.
Collection of the information of the boundaries are needed in the first step. Thus a whole image is needed in the first step, otherwise the wrong boundaries could exist during the process.
A simple example is shown in Figure-4.4. If only part of the image is concerned (in Figure-4.3b), this line can not be confirmed as part of the boundary, because without the information of the whole, this line could be anything.

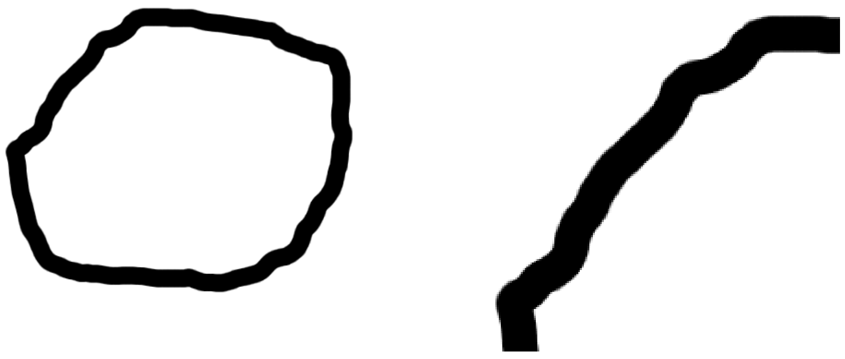

(a) original image **(b)** part of the image

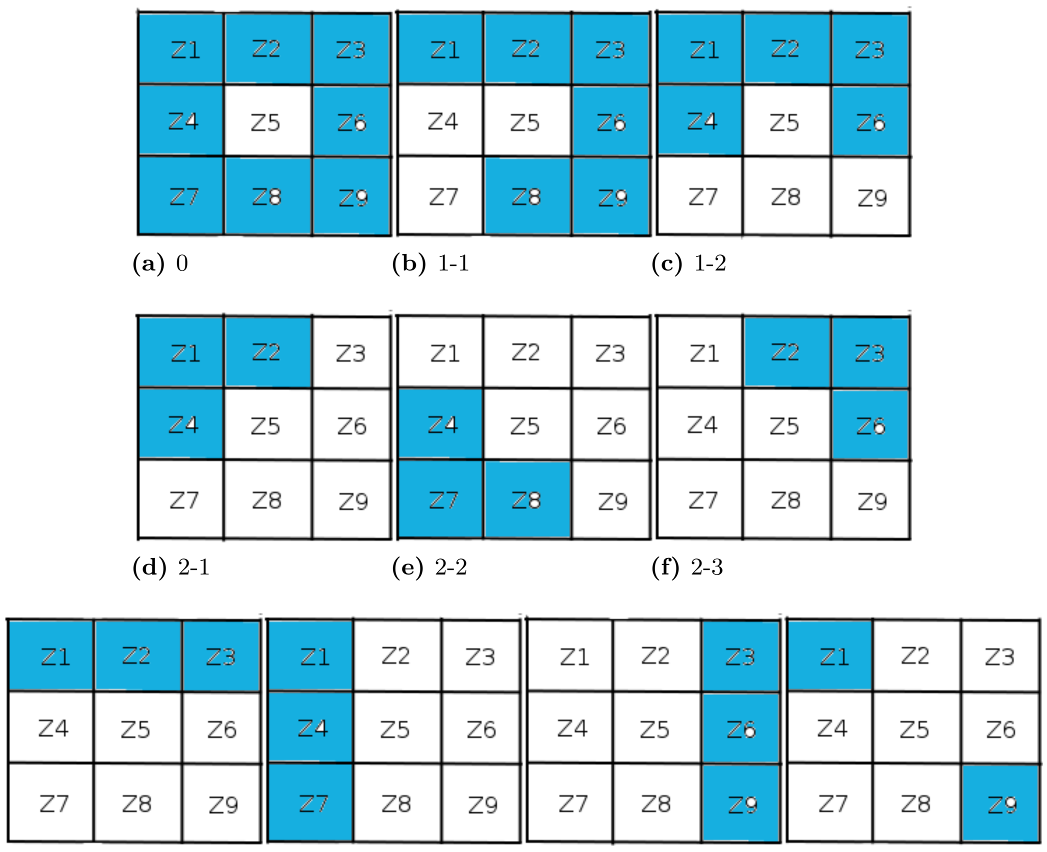
Figure 4.3: Boundary Example

The traditional hole filling method can not be applied here. An alternative method was used to do the same job.
The outcome was not perfect, but it worked.
A set of rules was introduced to decide whether a pixel is " in the hole filling area". A 3×3 filter was used in this case.

- The optimal situation is shown in Figure-4.4a, and the operation is very simple : filling Z_5 pixel without any doubt.
- The less optimal situation is that one of the 4 edges is missing , as shown in Figure-4.4b or in Figure-4.4c. Under this condition the center of the matrix can also be assigned with a valid value.
- When there are two edges missing, the center of the matrix can also be filled. Assume that this condition is at the intersection point of two edges. But the assumption could be wrong , thus the operation may cause some bias. Not all of the situation are considered as valid . This operation is not be applied on all of the corners.
  For example, the situation like Figure-4.4d is valid, because the scanning process is from top-left side to the right-bottom side. When this situation is encountered, in most cases the area in the boundary of the top-edge and left-edge is filled. To the contrary, as shown in Figure-4.4e , the center pixel resides most likely outside the area of the boundary, because this situation happens most outside the right-top area. Area between left-edge and bottom-edge could be filled through other way The situation shown in Figure-4.4f is also not suitable to fill.
- The next situation is when there is only one edge left. Just as mentioned, the bottom edge is not required in this algorithm. The top-edge, left-edge and the right edge are

already enough. Although a single right edge is rare, it was also included. Three types of the filters are shown in Figure-4.4g,Figure-4.4h,Figure-4.4i.

- The last situation is the diagonal situation. The diagonal line between Z_1 and Z_9 was chosen . The other one is not included because of the reason mentioned above. Figure-4.4j shows this situation.



(a) 0 **(b)** 1-1 **(c)** 1-2

(d) 2-1 **(e)** 2-2 **(f)** 2-3

(g) 3-1 **(h)** 3-2 **(i)** 3-3 **(j)** 4

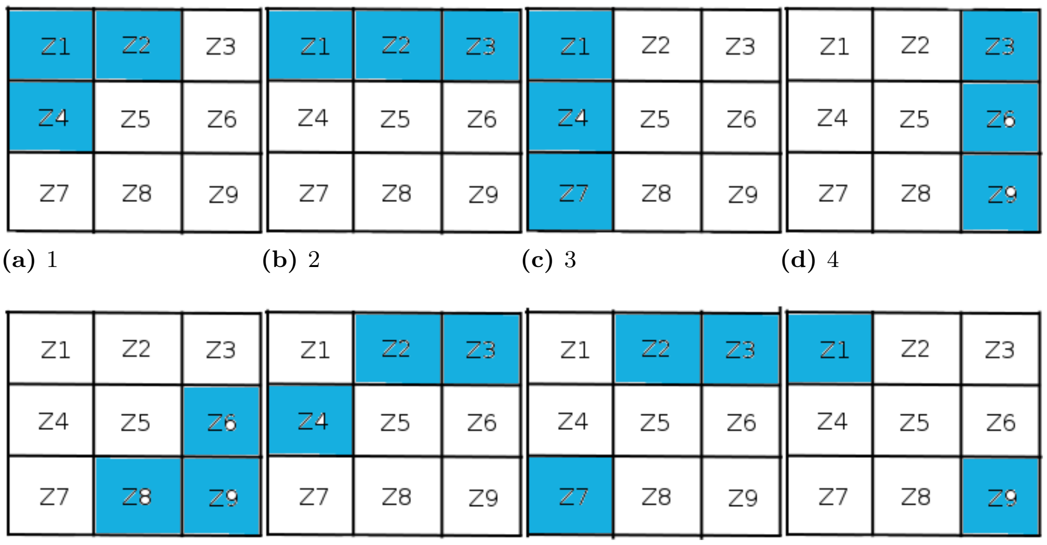
Figure 4.4: Boundary Example

In order to improve the performance, checking all pixels on one side was not necessary. In a 3×3 matrix, an edge contains only 3 pixels. The head and end pixels are enough to represent an edge. Thus these two were chosen. It is also easy to find a situation with 2 edges or 1 edge which can represent the situation with 3 or 4 edges. Then the simplified filters are shown in Figure-4.5.

For a 3×3 filter 2 rows and 3 pixels are needed. The input of this filter is the result of the dilation operation.

The output is the value of centered pixel.

The waiting time can be reduced from waiting for 1280 rows to only 2 rows.



(a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6 (g) 7 (h) 8

Figure 4.5: Filters used

4.2.4 Erosion Method

This method worked almost the same as the dilation method.
As the opposite method to dilation, a 7×7 filter was applied. Thus 6 rows and 7 pixels should be saved for temporary usage.
The input was the result of the hole filling operation.
The output was the value of the centered pixel.
The waiting time was also reduced from waiting for 1280 rows to only 6 rows.
To speed up this function, the similar scanning method was also introduced as in the dilation method: When a "blank" pixel was encountered, the result was directly set to 0 without scanning all values in the 7×7 matrix.

4.2.5 Alternative Approach to Regionprops Method

The Matlab built-in regionprops function can not be used here, the reason for dropping this function is that it needs the whole image as the input.
The following method was introduced in [5] and [6]. Assume that in region D a two-dimensional image intensity function $f(x, y)$ is defined. Parameters x and y are the column value and row value of the corresponding pixel. Then the $(r + c)$th order geometric moments of $f(x, y)$ can be expressed in

$$M_{c,r} = \int \int_D f(x, y) x^c y^r dxdy$$

In our case the filtered image is a binary image, which means that the value of a pixel is either "0" or "1". Thus the integral becomes addition, and the equation is changed to

$$M_{c,r} = \sum_D x^c y^r$$

Then two coordinates are introduced : $x_t = \frac{m_{10}}{m_{00}}$ and $y_t = \frac{m_{01}}{m_{00}}$. These two are called centroid coordinates. Image translation by vector $(-x_t, -y_t)$ can produce a centered image. When computed with respect to these object centroids the moment function can be modified as

$$\mu_{c,r} = \sum_D (x - x_t)^c (y - y_t)^r$$

A droplet shape can be represented by an ellipse who's center has the centroid coordinates of the droplet. A covariance matrix $[\begin{smallmatrix}\mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02}\end{smallmatrix}]$ can be built based on the second-order central moments. In the single-pass method, all of the second-order moments can be determined in this way: counting the number "N" of pixels that have the value "1" in the valid area where the droplet resides. These 3 expressions can be used:

$$\mu_{20} = \frac{1}{N}[\sum_{i=1}^{N} x_i^2 - \frac{1}{N}(\sum_{i=1}^{N} x_i)^2]$$

$$\mu_{02} = \frac{1}{N}[\sum_{i=1}^{N} y_i^2 - \frac{1}{N}(\sum_{i=1}^{N} y_i)^2]$$

$$\mu_{11} = \frac{1}{N}[\sum_{i=1}^{N} x^2 y^2 - \frac{1}{N}(\sum_{i=1}^{N} x_i)(\sum_{i=1}^{N} y_i)]$$

In order to save the temporary result , variables like sum of x_i (sum_x), sum of y_i (sum_y), , sum of square of x_i (sum_x_square), and sum of square of y_i .etc can be saved. The number "N" of valid pixels is represented by variable sum_one . The eigenvalues of this matrix can be found by using the equation:

$$\lambda_{l,s} = \frac{\mu_{20} + \mu_{02} \pm \sqrt{(\mu_{20} + \mu_{02})^2 + 4\mu_{11}^2}}{2}$$

The semi-axes a and b can be defermined by

$$a = 2\sqrt{\lambda_l}$$

$$b = 2\sqrt{\lambda_s}$$

These two semi-axes can be used in the equation of the ellipse

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

When the whole image is processed , the result can be calculated in the following sequence:

$$\mu_{20} = \frac{sum\_x\_square - \frac{sum\_x^2}{sum\_one}}{sum\_one};$$

$$\mu_{02} = \frac{sum\_y\_square - \frac{sum\_y^2}{sum\_one}}{sum\_one};$$

$$\mu_{11} = \frac{sum\_xy - \frac{sum\_x \times sum\_y}{sum\_one}}{sum\_one};$$

$$tr\_\mu = \mu_{20} + \mu_{02};$$

$$det\_\mu = \mu_{20} \times \mu_{02} - \mu_{11}^2;$$

$$\lambda_l = \frac{tr\_\mu + \sqrt{tr\_\mu^2 - 4 \times det\_\mu}}{2};$$

$$\lambda_s = \frac{tr\_\mu - \sqrt{tr\_\mu^2 - 4 \times det\_\mu}}{2};$$

$$r\_max = 2 \times \sqrt{\lambda_l};$$

$$r\_min = 2 \times \sqrt{\lambda_s};$$

$$area = \pi \times r\_max \times r\_min;$$

5 Results and Conclusion

5.1 Algorithm Test

In this section the algorithm was applied on some regular shapes and the effect is also discussed . The algorithm was applied on the following regular shaples : a triangle, a rectangle, a circle and an ellipse.
Area error ratio was used to check the result. This error ratio was determined by using equation

$$\frac{|area_a - area_m|}{area_m} \times 100\% \tag{5.1}$$

where $area_a$ was the result of the algorithm, $area_m$ was the area determined by the constructing program.

5.1.1 Triangle Test

The first to be tested shape was the triangle. The result is shown in Figure-5.1. This triangle image was created in Matlab . The used code is as follows:

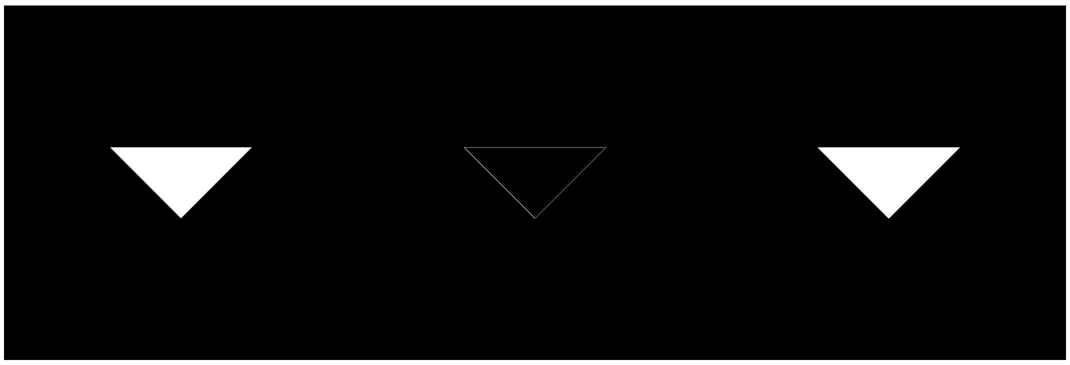
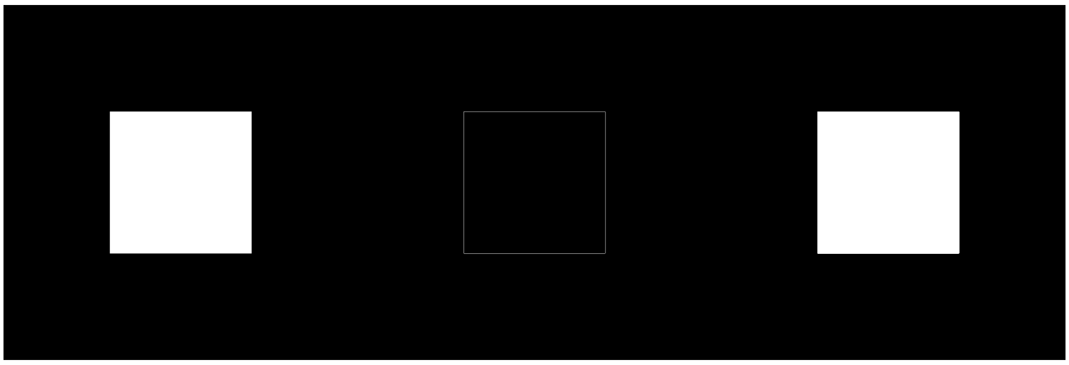

(a) test image **(b)** filtered by sobel filter **(c)** applied with the algorithm

Figure 5.1: Result on a Triangle



(a) test image **(b)** filtered by sobel filter **(c)** applied with the algorithm

Figure 5.2: Result on a Rectangle

```
A = eye(200,200);
B = A(:, end:-1:1);
C = [A,B];
C(1,:)=1;
C = imfill(C);
C = C * 255;
img_tri = zeros(1000,1000);
img_tri (401:600, 301:700) = C;
figure,imshow(img_tri);
```

The area of the triangle could also easily be calculated : $(200 \times 2 \times 200 \div 2 + 200 \times 2 = 40400$.
The algorithm returned the area value : 40599.
The error ratio was determined by Equation-5.1 : 0.49% .

5.1.2 Rectangle Test

The second one was rectangle.
The creation of rectangle was simpler than triangle. A rectangle shape could easily be obtained by setting the edges . The result is shown in Figure-5.2.
According to the Matlab code , the area of the rectangle was $400 \times 400 = 160000$.
The algorithm got a result of 160796 .
The error ratio was determined by Equation-5.1 : 0.5%

5.1.3 Circle Test

In this section the algorithm was applied on the circle shape.
The Matlab code for generating the circle [2] is as follows:

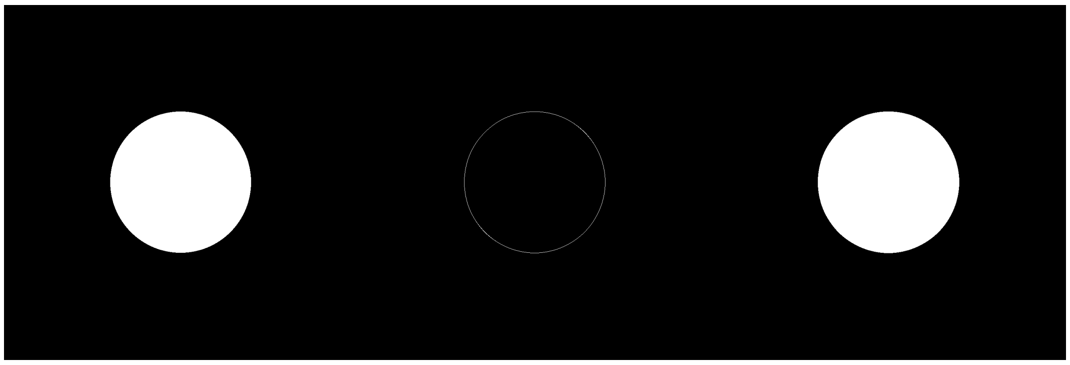

(a) test image **(b)** filtered by sobel filter **(c)** applied with the algorithm

Figure 5.3: Result on a Circle

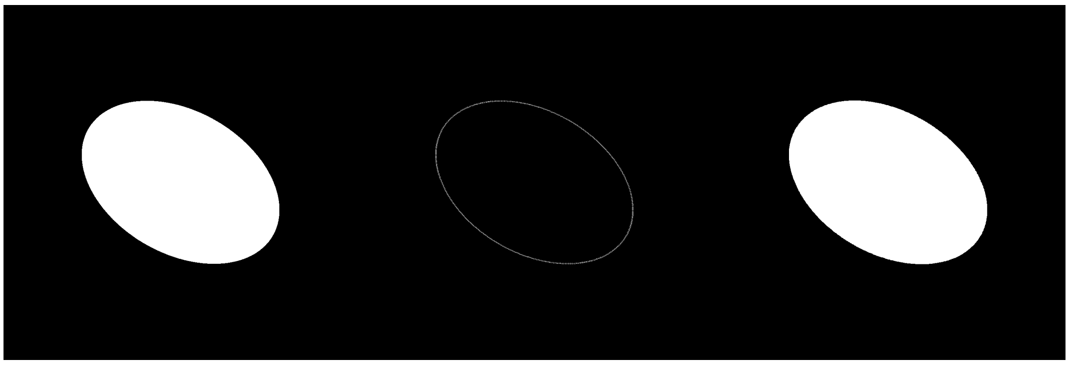
```
texsize = 400;
[x y]  = meshgrid(1:texsize, 1:texsize);
temp   = sqrt((x-texsize/2+.5).^2 + (y-texsize/2+.5).^2)> texsize/2-1;
temp = (1 - temp)*255;
img_cir = zeros(1000,1000);
img_cir(301:700,301:700) = temp;
```

And the result is shown in Figure-5.3. The area of the circle was $\pi \times 200^2 = 125664$.
The result of the algorithm was 125008.
Refering to 5.1, the error ratio was 0.52% .

5.1.4 Ellipse Test

The last test was on an ellipse.
The Matlab code for generating an ellipse with a major-axislength of 300 and minor-axislength of 200 is shown as follows:

```
a = 300;
b = 200;
rot_ang = (4/12) * pi ;
theta = 0:0.01:2*pi  ;
x_tmp = a * cos(theta);
y_tmp = b * sin(theta);
xx = cos(rot_ang) * x_tmp - sin(rot_ang) * y_tmp;
yy = sin(rot_ang) * x_tmp + cos(rot_ang) * y_tmp;
xx = round(xx) + 500;
yy = round(yy) + 500;
```



(a) test image **(b)** filtered by sobel filter **(c)** applied with the algorithm

Figure 5.4: Result on a Ellipse

shape	size	calculated size	error ratio
triangle	40400	40599	0.49%
rectangle	160000	160796	0.5%
circle	125664	125008	0.52%
ellipse	188496	190058	0.83%

Table 5.1: Algorithm Result

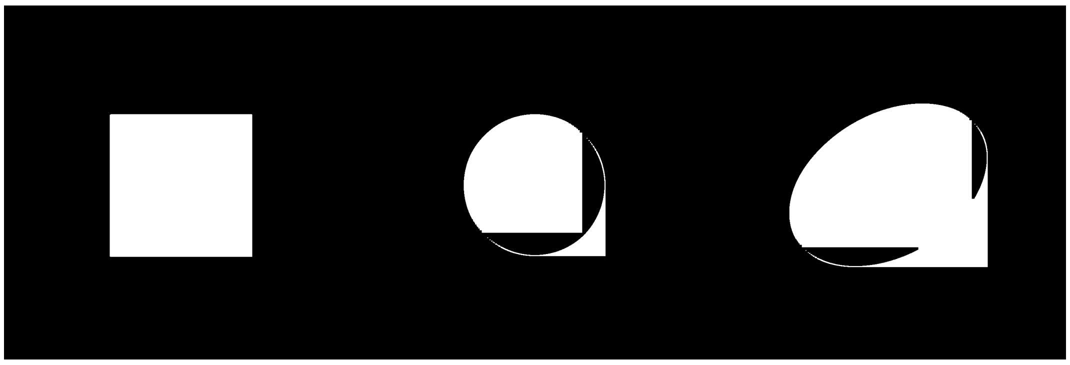
```
img_ell1 = zeros(1000,1000);
img_ell1( sub2ind( size(img_ell1), xx', yy')) = 255;
img_ell1 = imfill(img_ell1);
```

The result image is shown in Figure-5.4. The area of the ellipse shape was $\pi \times 300 \times 200 = 188496$.
The result of our algorithm was 190058.
The error ratio was calculated by Equation-5.1 : 0.83% .

The result indicated that the algorithm worked perfectly on the regular shapes.

5.1.5 Conclusion

Table - 5.1 shows the test result on some regular shapes. The correspondent tests can be found im Section-5.1. These results have only very small deviation compared with the original shapes. All of the error ratio were less than 1%. The algorithm worked well.



(a) Rectangle (b) Circle (c) Ellipse

Figure 5.5: Test Result

5.2 Single-Pass Algorithm Test

5.2.1 On regular Shapes

As in Section-5.1 the algorithm was tested on a rectangle, a circle, and an ellipse.
Because the mean filter was used in the algorithm, a thin edge could be just eliminated by mean filter, so the shape constructing method was modified to enlarge the edges. The modified methods can be found in A.1.1, A.1.2 and A.1.3. The filtered result is shown in Figure-5.5. Some parts of the edges were not connected , which affected the final result.
But the feature of our algorithm could be seen clearly in the result: The edges of the shape could be detected and most part of the pixels in the hole filling area were filled. The downside of the algorithm was that the right-bottom side of the shape was also filled.

5.2.2 Single-Pass Algorithm on the Droplet Image

After the single-pass algorithm has been applied on the original image, The result in Figure-5.6a was obtained.

Many noise pixels have been generated, and the pixels in the droplet was not fully set. Further modification was needed.
To eliminate the noise , 2 methods were used here:

- Raise the threshold of sobel filter on the area outside the shape of the droplet. The original threshold of the sobel filter was 7.
- Use mean filter to erase original noise pixels.

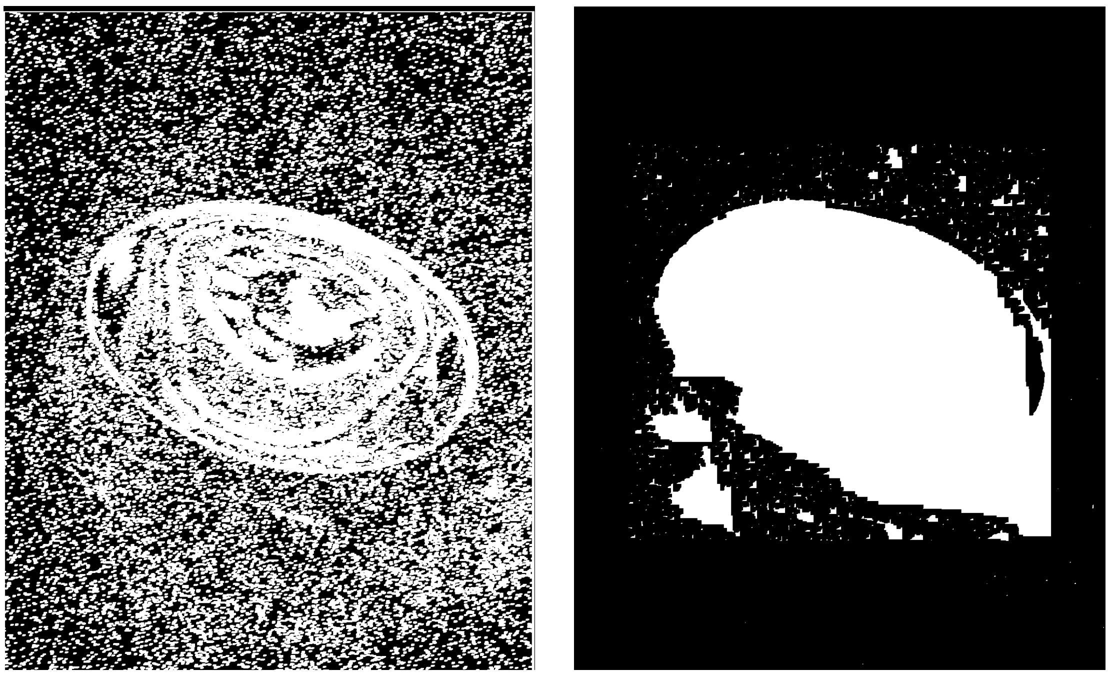

(a) first try (b) second try

Figure 5.6: Single-Pass Algorithm Result

To enhance the algorithm effect in the droplet, the threshold of sobel filter in this area was modified. According to the experiments, the modification has been done as follows:

- About the boundaries of the threshold changes: 10% length of from each side was chosen.
- About the values of the threshold: outside the droplet area it was set to 9, and within the droplet area it was set to 5 .

After the modification the result was as shown in Figure-5.6b.
This time the result was better.
The only problem left was that pixels were added on right-bottom side, and on the left-botteom side the noise area was enlarged. This problem was caused by the mean filter and the hole filling operation, and it was difficult to solve. If the threshold in the mean filter was increased, some parts of the droplet was missing. If some filters in the hole filling operation were removed, the shape of the droplet could not be obtained. The outcome of the modified algorithm was :

- Major-Axislength : 880.2
- Minor-Axislength : 653.2
- The area is 451642.5

When the Matlab built-in function regionprops was applied on the output image, the corresponding three numbers were: 878.7, 652.8, 371273. The Major- and Minor-Axislength were almost the same , The problem went with the area size. The algorithm simply multiplied the outcome of Major- and Minor-Axislength. To the contrary, the regionprops function counted how many pixels were in the droplet.
Two big blocks of noise at the left bottom of the droplet took responsibility for the problem. Another reason to explain the result difference between the original algorithm and the single-pass algorithm was that the hole filling method filled the right bottom of the droplet. One possible method to modify this result was calculating the difference between an ellipse and an rectangle. Here a was used to express the the half length of major axis, and b was used to express the half length of the minor axis. The area difference was regarded as

$$2a \times 2b - \pi \times a \times b = a \times b \times (4 - \pi);$$

The difference was about $(4 - \pi)$.
In our case only right-bottom side had the problem. When only a quarter in an ellipse was filled as a rectangle, the area difference could be calculated as

$$\frac{a \times b \times \pi}{a \times b \times \pi \times \frac{3}{4} + a \times b} = \frac{ab\pi}{ab(1 + \frac{3\pi}{4})}$$

The area difference is

$$\frac{\pi}{(1 + \frac{3\pi}{4})}$$

Thus the axis difference could also be obtained through multiplying $\sqrt{\frac{\pi}{(1+\frac{3\pi}{4})}}$. The axis result was adjusted as follows:

$$880.2 \times \sqrt{\frac{\pi}{(1 + \frac{3\pi}{4})}} = 851.6$$

$$653.2 \times \sqrt{\frac{\pi}{(1 + \frac{3\pi}{4})}} = 631.9$$

The droplet image processed with sobel filter had a result of 803.6 and 513.6, as described in Section-3.3.4.
The error ratio were

$$|851.6 - 803.6| \div 803.6 = 5.97\%$$

$$|631.9 - 513.6| \div 513.6 = 23.0\%$$

The area value obtained by the algorithm was adjusted as follows:

$$451642.2 \times \frac{\pi}{1 + 3\pi/4} = 422763$$

and the result with the regionprops method was adjusted as follows:

$$371273 \times \frac{\pi}{1 + 3\pi/4} = 347532$$

The original droplet image with sobel filter had a result of 319301 , as described in Section-3.3.4. Then the error ratio of the area were:

$$|422763 - 319301| \div 319301 = 32.4\%$$

$$|347532 - 319301| \div 319301 = 8.8\%$$

The error ratio of the algorithm result was very big. One reason was that the noise blocks on the left-bottom had negative effect on the feature-extracting method. Another reason was that the filtered object was not a regular ellipse, but the feature-extracting method treated the filtered object as an ellipse and just simply calculated the area with the ellipse area equation $a \times b \times \pi$. The big error ratio of the minor axis had proven the reasons. The result from the regionprops function showed the actual area result. This result had an error ratio of 8.8% .

5.2.3 Summary of Algorithm Requirements

In this section the needed memory space and saved time were discussed.
All space these methods needed were:

- sobel method: 2 rows + 3 pixels
- mean filter : 2 rows + 3 pixels
- dilation method: 6 rows + 7 pixels
- hole filling : 2 rows + 3 pixels
- erosion : 6 rows + 7 pixels
- area calculation: 6 variables

That was

$$(2 \times 3 + 6 \times 2) \times 1024 + (3 + 3 + 7 + 3 + 7 + 6) = 18461$$

units of memory space.
Each step in the original algorithm required a buffer size of $1024 \times englisch280 \times 1 = 1310720$ units. This buffer was repeatedly used. Thus the memory requirement difference could be calculated as

$$18461 \div (1024 \times 1280) \times 100\% = 1.4\%$$

The memory usage was significantly cut down. And the reduction of the processing time was also significant. Since the first few lines have been read, all the processing methods could be used parallelly.

6 Summary

In this article some generally used methods for processing images were introduced. The features of each method were compared. The effects of these methods on the droplet image have also been discussed.. After that the different composition of these methods have also been discussed. On the processed image a function for calculating the properties of the droplet was introduced.
Then a composited algorithm was built as

- smooth the image, or do nothing.
- detect edges.
- do morphological operations to connect the isolated pixels on edges.
- extract features of the droplet.

The algorithm could extract the droplet from the original image and also returned a reliable values for the properties(in Section-3.3.2). As discussed in Section-5.1.5, this algorithm had less than 1% error ratio when it was applied on regular shapes.
Then the algorithm was modified in order to reduce the memory usage and the processing time. A new hole filling method and a new droplet feature extracting method were introduced. The modified algorithm didn't work as good as the algorithm which we have found at first. Section-5.2.2 indicated that the major axis result had an error ratio of 5.97% , but the minor axis result had an error ratio of 23%. Thus the result of the area had an error ratio of 32.4%. Although the actual error ratio which had been calculated by Matlab built-in function regionprops was 8.8%, this error ratio not was not able to be achieved by the modified algorithm. Further consideration is needed for the modified algorithm.

A Appendix

A.1 Modified Shape Constructing Method

A.1.1 Rectangle

```
A = ones(400,400)*100;
A(1:end, 1) = 168;
A(1, 1:end) = 168;
A(end, 1:end) = 168;
A(1:end, end) = 168;
img_rect = zeros(1000,1000);
img_rect (301:700, 301:700) = A ;
```

A.1.2 Circle

```
texsize = 400;
[x y]  = meshgrid(1:texsize, 1:texsize);
temp   = sqrt((x-texsize/2+.5).^2 + (y-texsize/2+.5).^2)> texsize/2-1;
temp = (1 - temp)*50;
img_cir = zeros(1000,1000);
img_cir(301:700,301:700) = temp;
clear x y temp;
texsize = 398;
[x y]  = meshgrid(1:texsize, 1:texsize);
temp   = sqrt((x-texsize/2+.5).^2 + (y-texsize/2+.5).^2)> texsize/2-1;
temp = (1 - temp)*50;
img_cir(302:699,302:699) = img_cir(302:699,302:699)+ temp;
texsize = 396;
[x y]  = meshgrid(1:texsize, 1:texsize);
temp   = sqrt((x-texsize/2+.5).^2 + (y-texsize/2+.5).^2)> texsize/2-1;
temp = (1 - temp)*50;
img_cir(303:698,303:698) = img_cir(303:698,303:698)+ temp;
```

A.1.3 Ellipse

```
a = 300;
b = 200;
rot_ang = (8/12) * pi ;
theta = 0:0.00005:2*pi  ;
x_tmp = a * cos(theta);
y_tmp = b * sin(theta);
xx = cos(rot_ang) * x_tmp - sin(rot_ang) * y_tmp;
yy = sin(rot_ang) * x_tmp + cos(rot_ang) * y_tmp;
xx = round(xx) + 500;
yy = round(yy) + 500;
img_ell1 = zeros(1000,1000);
img_ell1( sub2ind( size(img_ell1), xx', yy')) = 100;
img_ell1 = imfill(img_ell1);
img_ell1( sub2ind( size(img_ell1), xx', yy')) = 200;
```

A.2 Part of Single-Pass Algorithm

A.2.1 Buffer Constructing

At the beginning of the program the buffers for temporary data in each step were constructed:

```
calc_img_4_sob = zeros(3 , len + 4);
calc_img_4_mean = zeros (3, len + 4);
calc_img_4_morph_d = zeros (7, len + 4);
calc_img_4_morph_f = zeros (3, len + 4);
calc_img_4_morph_e = zeros (7, len + 4);
```

When a row has been processed, these buffers were reconstructed. The first row was erased and a new row was added at the bottom of the matrix.

```
calc_img_4_sob = [ calc_img_4_sob(2:end, :) ; zeros(1, len + 4)] ;
calc_img_4_mean = [ calc_img_4_mean(2:end, :) ; zeros(1, len+4) ];
calc_img_4_morph_d = [ calc_img_4_morph_d(2:end, :) ; zeros(1,len+4) ] ;
calc_img_4_morph_f = [ calc_img_4_morph_f(2:end, :) ; zeros(1,len+4) ] ;
calc_img_4_morph_e = [ calc_img_4_morph_e(2:end, :) ; zeros(1,len+4) ] ;
```

A.2.2 Sobel Filter

In the code, “j” was the current column index.

```
calc_img_4_mean(3,j+1) = abs(sum(calc_img_4_sob(1,j:j+2)) ...
  - sum(calc_img_4_sob(3,j:j+2)) + calc_img_4_sob(1,j+1) ...
  - calc_img_4_sob(3,j+1) + sum(calc_img_4_sob(:,j)) ...
  - sum(calc_img_4_sob(:,j+2)) + calc_img_4_sob(2,j) ...
  - calc_img_4_sob(2,j+2));
if (i>sob_thresh_region_v && i<hei-sob_thresh_region_v ...
  && j>sob_thresh_region_h && j<len-sob_thresh_region_v)
  sob_thresh = sob_thresh_base - sob_thresh_diff ;
else
  sob_thresh = sob_thresh_base ;
end
if (calc_img_4_mean(3,j+1) <= sob_thresh)
  calc_img_4_mean(3,j+1) = 0;
else
  calc_img_4_mean(3,j+1) = 1;
end
```

A.2.3 Hole Filling

“calc_img_4_morph_f” is the input matrix for hole filling method. The current column index is j.

```
if ( (calc_img_4_morph_f(2,j-2)>0) || ...
        ((calc_img_4_morph_e(end,j-3)>0) && ...
        (sum(calc_img_4_morph_e(end-1,j-3:j-1)>1)) ) || ...
        (sum(sum(calc_img_4_morph_f(:,j-3:j-1)) >3)) || ...
        ( calc_img_4_morph_e(end-1,j-3)+calc_img_4_morph_e(end-1,j-2)+ ...
         calc_img_4_morph_e(end,j-3)+calc_img_4_morph_f(3,j-3)+ ...
         calc_img_4_morph_f(3,j-2) > 2) || ...
        (sum(calc_img_4_morph_f(3,j-3:j-1)) + ...
         sum(calc_img_4_morph_f(2:3,j-1) >2)))
    calc_img_4_morph_e(end,j-2) = 1 ;
```

Bibliography

[1] Gimp documentation. http://docs.gimp.org/en/index.html, 2012. (Cited on pages 16, 17, 18 and 19)

[2] Winston Chang. Matlab cookbook. http://wiki.stdout.org/matlabcookbook/Presenting%20visual%20stimuli/Quickly%20drawing%20many%20images%20or%20shapes/, 2009. (Cited on page 48)

[3] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing.* Pearson Pretice Hall, 3 edition, 2008. (Cited on pages 9, 10, 11, 12, 13, 14, 15 and 16)

[4] The Mathworks Inc. Matlab help. http://www.mathworks.de/de/help/documentation-center.html, 2012. (Cited on page 35)

[5] M Klaiber, S Ahmed, Z Wang, L Rockstroh, Y Gera, and S Simon. Online image analysis of spray processes based on a reconfigurable embedded system. *10. Workshop ueber Sprays, Techniken der Fluidzerstaeubung und Untersuchungen von Spruehvorgaengen, Berlin, Germany*, May.2012. (Cited on page 44)

[6] J Laackmann, S Ahmed, R Sedelmayer, M Klaiber, W Pauer, S Simon, and H.-U.Moritz. Investigation of polymerization and drying of polyvinylpyrrolidone in an acoustic levitator using a smart camera for online process measurement. *ICLASS 2012*, Sep.2012. (Cited on page 44)

[7] D Marr and E Hildreth. Theory of edge detection. *Proc.R.Soc.Lond.*, 1980. (Cited on page 12)

All links were last followed on November 27, 2012.

Declaration